Universidad Autónoma de Querétaro
Facultad de Informática
Maestría en Ciencias Computacionales

DESARROLLO DE ALGORITMOS PARA LA PLANEACIÓN Y OPTIMIZACIÓN DE RUTAS DE UN ROBOT MÓVIL EN AMBIENTES CON OBSTÁCULOS

TESIS

Que como parte de los requisitos para obtener el grado de Maestro en Ciencias Computacionales

Presenta:

ERIC FRANCISCO SALINAS GONZALEZ

Dirigido por:

Dr. Marco Antonio Aceves Fernández

SINODALES

Dr. Marco Antonio Aceves Fernández
Presidente

Dr. José Emilio Vargas Soto
Secretario

Dr. Efrén Gorrostieta Hurtado
Vocal

Dr. Jesús Carlos Pedraza Ortega
Suplente

Dr. Juan Manuel Ramos Arreguin
Suplente

M.C. Ruth Angélica Rico Hernández
Directora de la Facultad

Dr. Irineo Torres Pacheco
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
FEBRERO 2012
México

RESUMEN

El problema de la planificación de trayectorias de robots móviles se ha estudiado intensivamente en el pasado, debido a que una de las principales motivaciones es la utilización de robots móviles para tareas peligrosas, costosas o imposibles para los seres humanos. Existen diferentes técnicas para resolver este problema.

Muchos algoritmos que construyen mapas en tiempo de ejecución, necesitan robots con suficiente memoria, poder de procesamiento o técnicas de sensores avanzados como sensores laser o video cámaras con extensivo procesamiento de imágenes. Pero, muchas veces es necesario utilizar robots limitados, el término limitado se refiere a muy poco poder de procesamiento, pequeña memoria y un sensor muy limitado, para esto existen varios algoritmos que mueven al robot de un lugar a otro sin la necesidad de un mapa. En esta tesis se presenta una metodología para la solución al problema de la planificación de trayectorias de un robot móvil. Primero, se generan varios ambientes al azar por el cual el robot móvil tiene que navegar. Estos ambientes son conocidos como mazes o laberintos, en la medida que el maze es de mayor tamaño se vuelve un reto encontrar la salida. Posteriormente se implementó el algoritmo de Dijkstra para obtener la ruta más corta. Obteniendo la ruta por donde el robot móvil navegara. Una vez obtenida la ruta más corta se implementó el algoritmo de Montecarlo localización también conocido como filtrado de partículas logrando que el robot navegara por una ruta previamente definida. Todos los algoritmos son implementados en un lenguaje de alto nivel MATLAB.

(***Palabras clave.*** robots móviles, ruta más corta, dijkstra, mazes, Planificación de trayectorias, montecarlo localización.)

SUMMARY

The problem of planning paths of mobile robots has been studied intensively in the past; this is because one of the primary motivations on this field is the possibility of using robots to perform tasks that are dangerous, expensive, or simply impossible for humans. There are different approaches to solve this problem.

Most algorithms that build maps at runtime need robots with adequate memory, processing power, or advanced sensor techniques, such as laser sensors or video cameras with extensive image processing. However, sometimes it is necessary to use limited robots; limited in terms of very little computing power, small amount of memory and a limited number of sensor. Therefore, there are a number of algorithms that move the robot from one place to another without needing a map. This study presents a methodology for the solution to the problem of path planning of a mobile robot. First, are generated randomly several environments for which the mobile robot has to navigate. These environments are known as mazes or labyrinths, when the mazes are extended it becomes a challenge to find the exit. Later on, Dijkstra's algorithm is implemented to find the shortest path, obtaining the path where the mobile robot will navigate. Once the shortest path was obtained, the algorithm Monte Carlo localization also known as particle filtering was implemented, making the robot navigate by the path previously defined. All algorithms are implemented in a high-level MATLAB language.

(***Key word.*** mobile robots, shortest path, dijkstra, mazes, planning trajectories, montecarlo localization.)

DEDICATORIA

A mi Esposa
Por su amor incondicional y estar en todo momento conmigo.

A mis Hijos
Por ser el tesoro más grande de mi vida.

A mis Padres
Por darme todo lo necesario para salir adelante.

A mis Hermanos
Por su ayuda incondicional.

AGRADECIMIENTOS

Agradezco al Dr. Marco Antonio Aceves Fernández por su contribución integral y su liderazgo que hicieron posible la realización de este trabajo.

Agradezco a mis sinodales por el tiempo empleado en la revisión de este trabajo.

Agradezco a mi tío el Lic. José Dolores González Ortiz por su ánimo y apoyo durante toda mi vida.

Agradezco a mis amigos, ya que cada uno de ellos es una inspiración a su manera y me ayudan a forjar dentro de mí la confianza que puede ser determinante.

Í N D I C E

Página

INDICE DE FIGURAS

INDICE DE TABLAS

Capítulo 1

Introducción

1. INTRODUCCION.

El problema de la planificación de trayectorias de robots móviles se ha estudiado intensivamente en el pasado, debido a que una de las principales motivaciones, es la utilización de robots móviles para tareas peligrosas, costosas o imposibles para los seres humanos. Aunque los robots móviles se utilizan para realizar tareas complejas el problema fundamental de la localización subyace en todas las aplicaciones de robots móviles, incluso cuando un mapa está disponible, no es un problema trivial. Realizar una tarea de navegación para un robot móvil significa recorrer un camino que lo conduzca desde una posición inicial hasta otra final, pasando por ciertas posiciones intermedias o sub metas. [6]

El entorno de trabajo en el cual un robot móvil realizará su tarea, puede considerarse como un conjunto de configuraciones en las cuales puede encontrarse el robot en un determinado instante de tiempo. Entre ellas existirá un subconjunto inalcanzable, al estar ocupado por los obstáculos del entorno.

En este trabajo de investigación se analizara y desarrollarán algoritmos para la planeación y optimización de rutas, obteniendo el camino más corto de un punto inicial a un punto final evitando obstáculos, que permitirán la navegación de un robot móvil en ambientes con obstáculos (conocidos como mazes o laberintos).

Navegar en un ambiente desconocido es una de las tareas más importantes que un robot autónomo tiene que resolver. [5] Existen diferentes técnicas para

resolver este problema, pero todas ellas poseen en común el llevar al robot a su destino de forma segura. Los objetivos principales de esta tesis son:

- Generar estructuras con obstáculos en un plano de dos dimensiones en donde el Robot móvil estará interactuando.
- Desarrollar metodologías para la planeación de rutas de un robot móvil, obteniendo el camino más corto de un punto inicial a un punto final evitando obstáculos.
- Analizar los algoritmos que permitan la navegación de un Robot móvil.
- Realizar la implementación de los algoritmos en un lenguaje de alto nivel *(en este caso* Matlab*)*.

La aportación de este trabajo es presentar un análisis detallado de los algoritmos que dan solución al problema de navegación de un robot móvil en ambientes desconocidos y que sirva como base para futuras investigaciones. Este trabajo está organizado de la siguiente forma: primero se presenta un marco teórico en donde se describe el problema de la navegación de los robots móviles así como las estructuras que se utilizaran durante el desarrollo del trabajo.

Otro tema importante a ser tratado en este trabajo es la obtención de la ruta más corta utilizando el algoritmo de Dijkstra. Después en la sección de metodología se analiza el algoritmo de Monte Carlo que se implementó para simular la navegación del robot móvil, se realizan varios experimentos en donde se muestra el comportamiento de los algoritmos.

Capítulo 2

Marco Teórico

2. MARCO TEORICO

2.1 Robots Móviles

El principal objetivo de la robótica es la construcción de máquinas capaces de realizar tareas con la flexibilidad, la robustez y la eficiencia que exhiben los humanos. Los robots son potencialmente útiles en escenarios peligrosos para el ser humano, o de difícil acceso [18].

En este sentido, los brazos robóticos que se emplean en las fábricas de coches para soldar y pintar, los robots móviles que se envían a Marte, o los que se utilizan para limpiar centrales nucleares son varios ejemplos de aplicaciones reales en las cuales se utilizan robots hoy día.

Existen diversos tipos de robots: con ruedas, con patas, brazos manipuladores, con orugas, de forma humanoide, de forma cilíndrica, etc. Sin embargo, la morfología no es una característica esencial, lo que identifica a cualquier robot es que combina en la misma plataforma a sensores, actuadores y procesadores. Los sensores miden alguna característica del entorno o propia (ej. cámaras, sensores de obstáculos, etc.). Los actuadores permiten al robot hacer algo, llevar a cabo alguna acción o simplemente desplazarse (ej. los motores). Los procesadores hacen los cálculos necesarios y realizan el enlace lógico entre sensores y actuadores, materializando el comportamiento del robot en el entorno en el cual se encuentra inmerso.

Muchos de los robots que se venden en la actualidad se compran como productos cerrados, programados por el fabricante e inalterables, por ejemplo el robot Pioneer 1. (Figura 1). Es un robot modular móvil que ofrece varias opciones, está

equipado con una biblioteca de navegación sofisticada desarrollada en el Stanford Research Institute (SRI).

Figura. 1. Robot móvil Pioneer 1 [21].

El robot móvil se caracteriza por realizar una serie de desplazamientos (navegación) y por llevar a cabo una interacción con distintos elementos de su entorno de trabajo (operación), que implican el cumplimiento de una serie de objetivos impuestos según sus especificaciones.

El robot móvil debe poseer una arquitectura que coordine los distintos elementos de a bordo (sistema sensorial, control de movimiento y operación) de forma correcta y eficaz para la realización de una misión.

Existen dos clases de sensores de robots; uno de ellos son los *sensores de rango* que proveen datos acerca de la distancia entre el robot y los objetos a su alrededor. El otro es el *sensor de visión* que provee datos acerca del entorno [19].

Los *sensores de rango* más comunes son sensores de contacto, sensores de ruido y sensores laser.

Los *sensores de contacto* dan información al robot acerca de si este está en contacto con otros objetos y que tanta presión está ejerciendo sobre el sensor de contacto. Los sensores de contacto son útiles para determinar si un robot ha sido golpeado o está siendo detenido por un objeto en el ambiente.

Los sensores *de sonido y sensores laser* proveen al robot con información acerca de la distancia a otros objetos. Ellos envían una señal y cuando la señal rebota en un objeto y regresa, el tiempo que tarda la señal en regresar es tomado para calcular la distancia.

Los sensores laser son más precisos que los sensores de sonido por que los rayos viajan rápidamente y son menos dispersos que los rayos de sonido. Así que, si un sensor de sonido y un sensor laser miden la distancia en la misma dirección, el sensor laser debe medir la distancia primero. Si el robot está en la acción de girar o de moverse hacia adelante el sensor laser debe de producir un resultado más exacto.

Los *sensores de visión* más comunes en un robot son cámaras. Los robots cámaras son utilizados para tomar imágenes periódicamente del entorno del robot o para monitorear continuamente el entorno de un robot por medio de un video en vivo.

Un video en vivo da más información al robot acerca del entorno continuamente, sin embargo, el procesamiento del video consume tiempo desde que envuelve cálculos extensos. Por lo que si un robot necesita información rápidamente es siempre más rápido trabajar con imágenes. La desventaja de imágenes es que proveen información discreta del ambiente. En el tiempo entre que las dos imágenes son tomadas, el ambiente puede cambiar drásticamente [16].

2.2 Problema de Navegación de un Robot Móvil

Se define navegación como la metodología que permite guiar el curso de un robot móvil a través de un entorno con obstáculos. Existen diversos esquemas, pero todos ellos poseen en común el afán por llevar el vehículo a su destino de forma segura. La capacidad de reacción ante situaciones inesperadas debe ser la principal cualidad para desenvolverse, de modo eficaz, en entornos no estructurados.

Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno a través de sus sensores, de modo que le permita crear una abstracción del mundo; la planificación de una trayectoria libre de obstáculos, para alcanzar el punto destino seleccionado; y el guiado del vehículo a través de la referencia construida. De forma simultánea, el vehículo puede interaccionar con ciertos elementos del entorno. Así, se define el concepto de operación como la programación de las herramientas de a bordo que le permiten realizar la tarea especificada.

Existen diferentes técnicas para resolver este problema. Muchos algoritmos que construyen mapas en tiempo de ejecución, necesitan robots con suficiente memoria, poder de procesamiento o técnicas de sensores avanzados. Pero, muchas veces es necesario utilizar robots limitados por razones de costo y espacio, el término limitado se refiere a muy poco poder de procesamiento, o memoria limitada, para esto existen varios algoritmos que mueven al robot de un lugar a otro sin la necesidad de un mapa [5].

Una de las técnicas más utilizadas para localización de robot móviles se conoce como Dead-Reckoning.

Dead-reckoning es una técnica de localización del robot. Como el robot se mueve a través de su ambiente, este tiene la distancia estimada y la dirección en el cual este ha viajado, así como un cambio en su orientación. Dead-reckoning utiliza la posición inicial y la información obtenida para determinar su posición en un tiempo determinado. Sin embargo, la información que el robot tiene acerca de sus movimientos no siempre corresponde exactamente a su movimiento actual. Dead-reckoning no considera la interacción entre el robot y su ambiente y no puede ser utilizado para la localización exacta del robot aunque el robot tenga información precisa de sus movimientos en el ambiente.

Dead-reckoning se basa en la posición inicial del robot para actualizar la posición actual. Si la posición actual no es conocida, el robot no tiene la manera de determinar la posición exacta en el ambiente.

Otra técnica de localización es la de filtro de Kalman un filtro en este contexto es un conjunto de ecuaciones matemáticas que son utilizadas para remover ruido en la información obtenida para predecir la posición del robot en el ambiente. El algoritmo del filtro de Kalman es un algoritmo estadístico para la solución del problema de seguimiento. Muchos de estos algoritmos asumen que la posición incierta de un robot y un sensor de información puede ser modelada por una distribución Gaussiana [8].

Los algoritmos que utilizan el filtro de Kalman asumen que el error en la posición del robot puede ser representado por una distribución normal, como se muestra en la Figura 2.

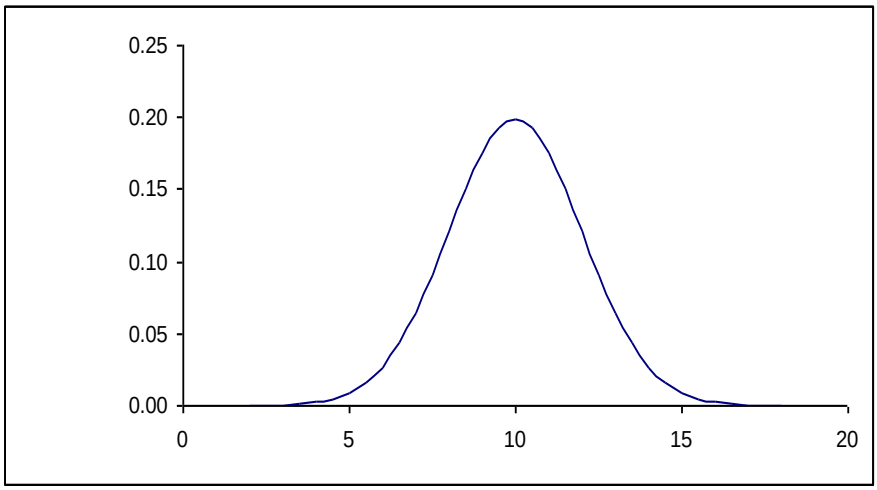

Figura 2. Distribución Gaussiana.

Una técnica diferente de localización es el algoritmo de localización de Markov. Este algoritmo representa el ambiente del robot como un conjunto de posiciones discretas L, a diferencia del filtro de Kalman, esta técnica requiere que el robot tenga un conocimiento previo acerca de su entorno, usualmente es un mapa. Sea $l \varepsilon L$, l = <*x,y,θ*>, donde *x* y *y* son las coordenadas del robot en el entorno y θ es la orientación[17].

El movimiento del robot es modelado por una probabilidad condicional, denotada por $p_a(l|l')$. $p_a(l|l')$ Denota la probabilidad que la acción del movimiento *a*, cuando se ejecuta l', lleva el robot a l. $p_a(l|l')$ Es utilizado para actualizar el Bel sobre el robot en movimiento. Donde $\widehat{Bel}(L_t = l)$ denota el Bel resultante para el tiempo t:

$$\widehat{Bel}(L_t = l) \leftarrow \sum_{l'} p_a(l|l') Bel(L_{t-1} = l') \qquad (1)$$

Sea s que denota un sensor de lectura y $p(s|l)$ la probabilidad de percibir a s en l. $p(s|l)$. Usualmente se refiere al mapa del entorno, desde este se especifican las

probabilidades de observación en las diferentes localidades en el ambiente. Cuando el sensor s, élBel es actualizado de acuerdo a la siguiente regla:

$$Bel(L_t = l) \leftarrow \frac{p(s|l)\widehat{Bel}(L_t=l)}{p(s)} \qquad (2)$$

En la ecuación 2, p(s) es un normalizador que asegura que el Bel suma 1 para todo l.

El algoritmo de localización de Markov es más exacto que el filtro de Kalman y puede ser utilizado para una localización global. Sin embargo la localización de Markov es difícil de implementar [16].

El algoritmo requiere que el entorno del robot sea divido sobre una cuadricula para asegurar la exactitud de los resultados. Así la localización exacta requiere un gran número de puntos para ser actualizados en cada iteración lo cual hace que la localización de Markov sea muy cara en tiempo y memoria.

Localización Monte Carlo (MCL) es una técnica estadística para la localización del robot basada en la localización de Markov. Como en la localización de Markov (MCL), requiere un mapa del entorno del Robot. Esto es un filtro de Bayes [17]. El algoritmo de Localización Monte Carlo es el que se desarrollará a detalle en el capítulo 3 de esta tesis.

2.3 Planeación y optimización de rutas de un Robot Móvil

El entorno de trabajo en el cual un robot móvil realizará su tarea, puede considerarse como un conjunto de configuraciones en las cuales puede encontrarse el robot en un determinado instante de tiempo. Entre ellas existirá un subconjunto inalcanzable, al estar ocupado por los obstáculos del entorno.

Se define una configuración *q* de un robot como un vector cuyas componentes proporcionan información completa sobre el estado actual del mismo. La localización del vehículo en un determinado instante de tiempo queda definida por la relación existente entre el sistema de coordenadas global en virtud del cual está definido todo el entorno de trabajo y su sistema de coordenadas locales asociado.

En un entorno de trabajo en donde dada una meta y un modelo de estados, un robot puede inferir sus rutas desde el estado actual a una meta por una transición recursiva de estados desde la meta. El objetivo es encontrar la ruta más corta desde todos los estados a una meta en un grafo dirigido.

Para obtener la ruta más corta de un vértice a otro vértice se utilizará el algoritmo de Dijkstra.

Este algoritmo determina el camino más corto dado un vértice origen al resto de los vértices en un grafo conexo y con pesos no negativos en cada arista. Su nombre se refiere a Edsger Dijkstra quien lo describió por primera vez en 1959 [22].

Definición: *Peso* es el costo de ir de un vértice a otro vértice en el grafo.

Se comienza con un grafo dirigido conexo sin ciclos G. A cada arista $e = (a, b)$ de este se le asigna un número real positivo llamado el peso de *e*, que se denota con

w(*e*) o w(a,*b*). Si x,y $\in$ V pero (x,y) $\notin$ E, se define que $w(x,y) = \infty$. (3)

Cuando se da un grafo G con las asignaciones de peso descritas como en la ecuación 3, se dice que el grafo es un grafo *ponderado*.

Dado G, para cada $e = (a,b) \in E$ se interpreta p(*e*) como la longitud de una ruta directa de a a *b*. Para cualesquiera dos vértices a,*b*$\in$ *V* se escribe (a,*b*) como la distancia (más corta) de a *a b*. Si no existe tal camino (en G) de a a *b* entonces se define$d(a,b)=\infty$, Para cualquier a $\in$ V, *d*(a,a) = 0.

Se fija ahora $v_0 \in V$. Entonces para cualquier $v \in$ V, se determina

1) $d(v_0,v)$
2) un camino simple dirigido de v_0 a v si $d(v_0,v)$ es finito.

Sea | *V* | = *n*. Para determinar la distancia más corta de un vértice fijo v_0 a los demás vértices de G, así como un camino simple dirigido más corto para cada uno de estos vértices, se aplica el siguiente algoritmo.

Paso 1: Se inicializa$i$ = 0 y $S_0 = \{v_0\}$. Se etiquetav_0 con (0, —) y cada $v \neq v_0$ con (∞,—)

Si n = 1, entonces V = { v_0} y el problema está resuelto.

Paso 2: Para cada $v \in \overline{S}_i$, se reemplaza, la etiqueta de v por la nueva etiqueta final ($L(v)$,y), donde

$$L(v) = \text{mín } \{L(v), L(u) + p(u,v)\}, U \in S_i \qquad (4)$$

Y y es un vértice en S_i que produce el $L(v)$ mínimo. Si se hace un reemplazo, esto se debe al hecho de que se puede ir de v_0 a v y recorrer una distancia más corta si se recorre un camino que incluye una arista (y,v).

Paso 3: Si cada vértice de S_i (para algún $0 \leq i \leq n-2$) tiene la etiqueta (∞, —) , entonces el grafo etiquetado contiene la información del camino más corto.

En caso contrario, existe al menos un vértice $v \in S_i$ que no está etiquetado como (∞ , —) y se realizan las siguientes tareas:

1) Se selecciona un vértice v_{i+1}, tal que $L(v_{i+1})$ sea mínimo. Puede haber varios de estos vértices cuyo caso se elige alguno de los posibles candidatos. El vértice v_{i+1} es un elemento de S_i que es el más cercano a v_0.

2) Se asigna S_i

3) Se actualiza i= i+1.

Si i = n – 1, el grafo etiquetado contiene la información del camino más corto. Se Regresa al paso 2.

Ejemplo: Se aplicara el algoritmo de Dijkstra al siguiente grafo ponderado T.

Para determinar la distancia más corta del vértice inicial (=v_0) a cada uno de los otros vértices de G.

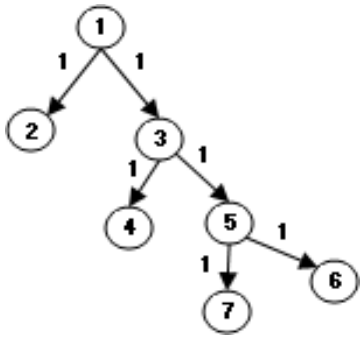

Figura 3. Árbol ponderado.

Paso 1: $S_0 = \{1\}$ $\overline{S}_0 = \{2,3,4,5,6,7\}$

Se Inicializa etiquetando el vértice 1 con 0 y los demás vértices con ∞.

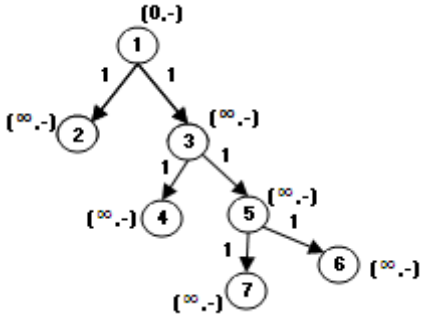

Figura 4. Árbol ponderado etiquetado

Paso 2:

$L(2) = \min(L(2), L(1) + p(1,2)) = \min(\infty, 0 + 1) = 1$

$L(3) = \min(L(3), L(1) + p(1,3)) = \min(\infty, 0 + 1) = 1$

$L(4) = \min(L(4), L(1) + p(1,4)) = \min(\infty, 0 + \infty) = \infty$

$L(5) = \min(L(5), L(1) + p(1,5)) = \min(\infty, 0 + \infty) = \infty$

.

.

.

$L(7) = \min(L(7), L(1) + p(1,7)) = \min(\infty, 0 + \infty) = \infty$

Paso 3: Se obtiene el vértice de valor mínimo. Se observa que los vértices 2 y 3 tienen el valor mínimo para este caso se toma el mayor vértice. Se etiquetan los vértices con (1,1) y se continúa con la siguiente iteración **paso 2**.

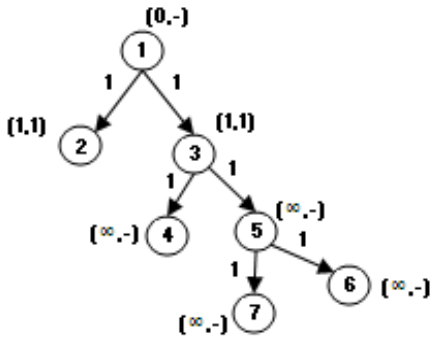

Figura 5. Árbol ponderado etiquetando el vértice 2 y 3 con (1,1).

Segunda iteración: $S_0 = \{1,3\}$ $\quad \overline{S}_0 = \{4,5,6,7\}$

L(4)= min (L(4), L(1) + p(1,4), L(3) + p(3,4)) = min (∞, 0 + ∞,1 + 1) = 2

L(5) = min (L(5), L(1) + p(1,5), L(3) + p(3,5)) = min (∞, 0 + ∞,1 + 1) = 2

L(6)= min (L(6), L(1) + p(1,6), L(3) + p(3,6)) = min (∞, 0 + ∞,1 + ∞) = ∞

L(7) = min (L(7), L(1) + p(1,7), L(3) + p(3,7)) = min (∞, 0 + ∞,1 + ∞) = ∞

Tercera iteración: $S_0 = \{1,3,5\}$ $\quad \overline{S}_0 = \{6,7\}$

L(6)= 3

L(7) = 3

Cuarta iteración: $S_0 = \{1,3,5,7\}$. El proceso termina y se obtiene la distancia más corta del vértice inicial a los demás vértices de T.

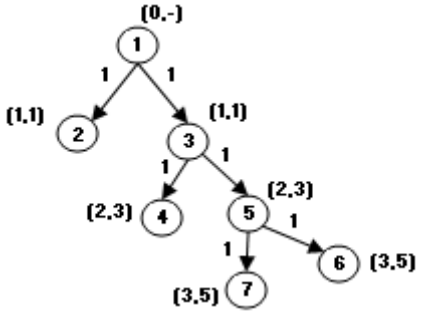

Figura 6. Árbol ponderado final.

1) d(1,3) = L(3) = 1
2) d(1,5) = L(5) = 2
3) d(1,7) = L(7) = 3

Dichos pasos del algoritmo de Dijkstra se muestran en el siguiente pseudocódigo

Pseudocódigo

Inicialización:

j = 1; F(1) = 0; F(i) = ∞, i ∈ {2 . . . , n}; U = C

Iteración:

Mientras (j ≠ n y F(j) <∞) **Hacer:**

Actualizar U : U = U\{j}

Actualizar F : F(i) = min{F(i), F(j) + D(j, i)}, i ∈ A(j) ∩ U

Actualizar j : j = argmin{F(i) : i ∈ U}

2.4 Modelado de ambiente con obstáculos

Navegar en un ambiente desconocido es una de más importantes tareas que un robot autónomo tiene por resolver. Para modelar el recorrido de un robot en ambientes desconocidos se utilizará en este caso una estructura geométrica llamada maze. [5]

Aunque el término de laberinto y maze se ha utilizado intercambiablemente existe una diferencia entre ellos. Los **laberintos** son aquellos que tiene un solo camino *único* al centro.

En el caso del **Maze** es un rompecabezas complejo formado de ramificaciones de pasajes en el cual se debe de encontrar una ruta.

Un maze tiene ciertas características que a continuación se describen:

Entrada: Es un punto en donde podemos acceder al interior del maze. Los mazes pueden tener más de un punto inicial pero normalmente no tiene más de uno.

Salida: Es un punto que se alcanza cuando se resuelve un maze. Los mazes pueden tener uno o más puntos finales pero normalmente no tiene más de uno.

Pared: es una barrera que no nos permite pasar cuando recorremos el maze.

Pasaje: Es un corredor en donde uno puede ir de un punto a otro punto. Un pasaje tiene paredes sobre ambos lados.

Ruta: Camino o dirección que se toma para conseguir una meta.

Solución: Es una ruta en un maze desde la entrada a la salida.

Entronque: Es un punto en donde dos o más pasajes se reúnen.

Callejón sin Salida: Es un pasaje que termina en un extremo de un bloque, lo que obliga a retroceder.

Pared exterior: una pared en el perímetro exterior del maze, que tiene un pasaje de un lado de este y un área que no es parte del maze del otro lado.

Grid: se refiere a dos o más conjuntos infinitos de líneas uniformemente espaciadas para ángulos en particular o la intersección de dichas líneas.

Celda: es un pasaje unitario los cuales son conectados en un grid para formar un maze.

Circuito Cerrado: es una ruta que se conecta con ella misma formando un ciclo.

Dimensión: Los mazes perfectos son bidimensionales.

Embaldosado: Los mazes perfectos tienen un embaldosado Ortogonal y se refiere a que es un grid rectangular en donde las celdas tienen pasajes de intersección en ángulo recto.

En la Figura 7. se muestran las características de un maze.

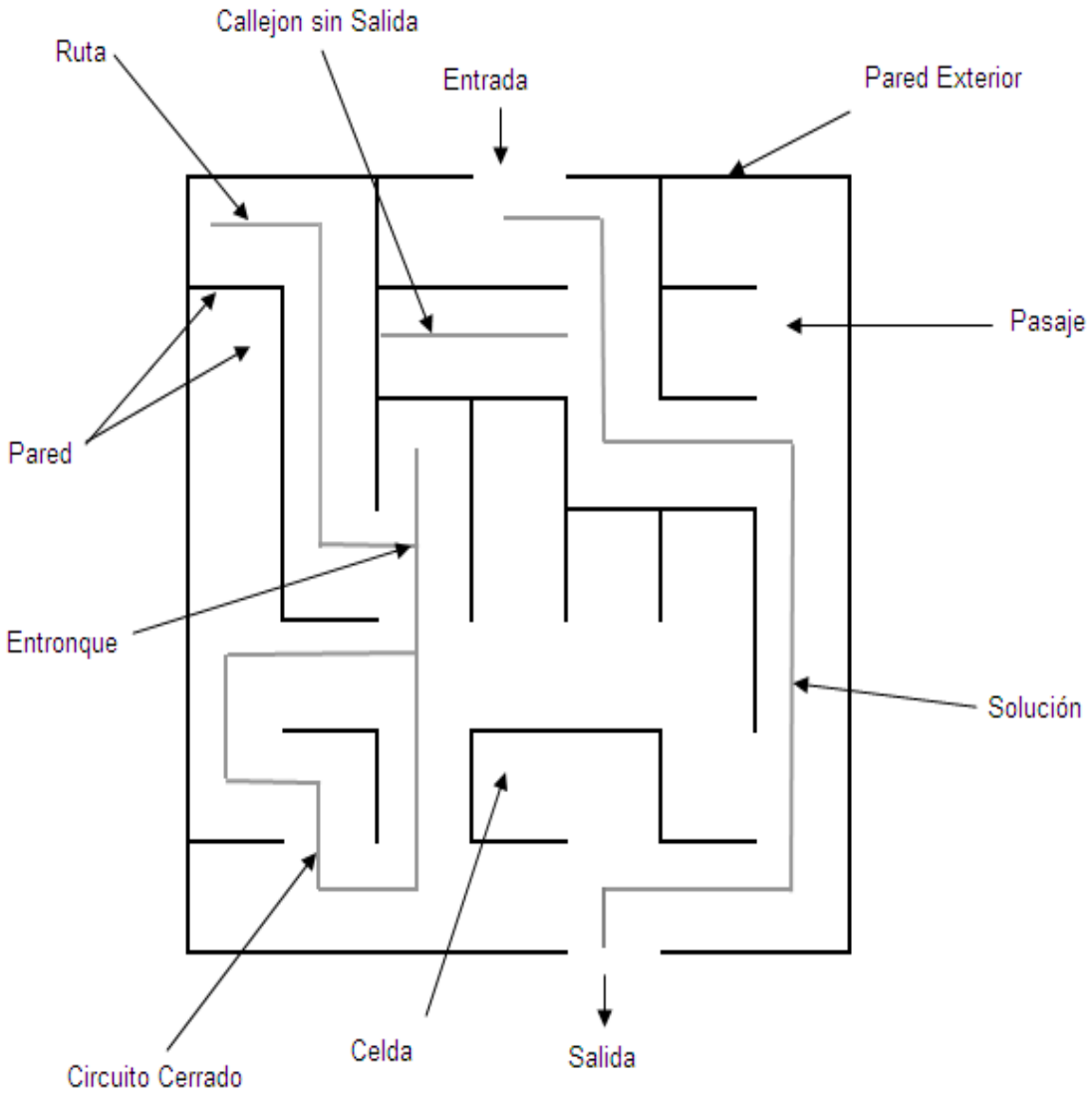

Figura 7. Características de un Maze.

2.5. El problema del Maze

El objetivo de este problema es que partiendo de una posición inicial (entrada del maze), se llegue a la posición final (salida del maze).

Para la solución del problema se utilizaran algunos algoritmos que se describirán en las secciones siguientes:

2.5.1. Generación de un Maze Perfecto

Un maze perfecto significa que no tiene ciclos o circuitos cerrados y no tiene zonas inaccesibles. También es llamado simplemente conectado. Desde cada punto, existe exactamente una ruta a cualquier otro punto.

Paso 1: Sea *m* una matriz de celdas de tamaño *n* y sea *w* el arreglo de paredes.

Inicializamos $m_{i,j}$= 0, así como w_i = 0.

Paso 2: Se construyen las paredes externas

$m_{1,j}$ y w_2 = 1

$m_{i,1}$ y w_3 = 1

$m_{n,j}$ y w_1 = 1

$m_{i,n}$ y w_4 = 1

Paso 3: Se selecciona al azar la entrada y salida.

Entrada = Random[1,n]

Salida = Random[1,n]

Paso 4: Adicionando paredes de tal manera que cada nuevo segmento de pared toca una pared en su lado final y tiene el otro lado final sin tocar una pared.

Para i = 0 **hasta** n-1

Para j=0 **hasta** n-1

/*Obtenemos el total de paredes de la celda con valor 0 */

Hacer

AgregarPared = False

contador = **TotalPared(**$m_{i,j}$**)**

/*Utilizamos la función random para obtener la pared que se Adicionara*/

pared = Random[1,contador]

Si Direccionpared **Entonces**

Si AgregaPared **Entonces**

Continuamos con la siguiente celda **y** AgregarPared = True

Mientras contador > 0 o AgregarPared = True.

Ejemplo: Aplicaremos el Algoritmo anterior para generar un maze perfecto.

Paso 1) Sea *n* = 5.

Paso 2) y 3) Construimos las paredes externas. Seleccionamos al azar la entrada y salida utilizando la función Random.

Entrada = Random[1,n]

Salida = Random[1,n]

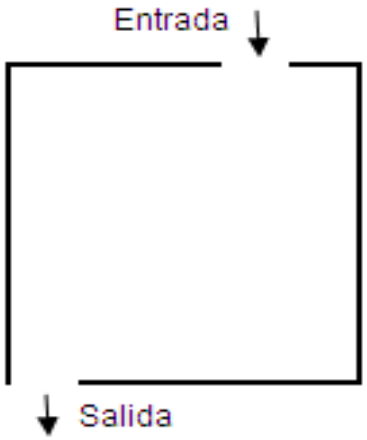

Figura 8. Paredes externas de un maze.

Paso 4) Adicionando paredes de tal manera que cada nuevo segmento de pared toca una pared en su lado final y tiene el otro lado final sin tocar una pared y que este segmento existente este dentro del maze.

Iteración 1

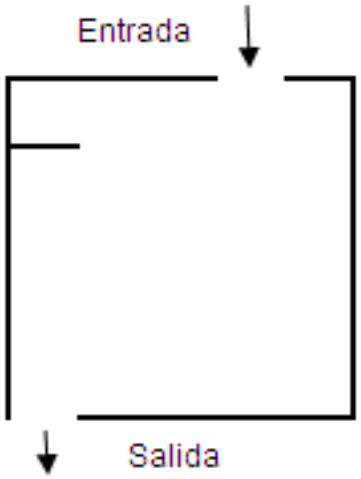

Figura 9. Pared interna de un maze

Iteración 25

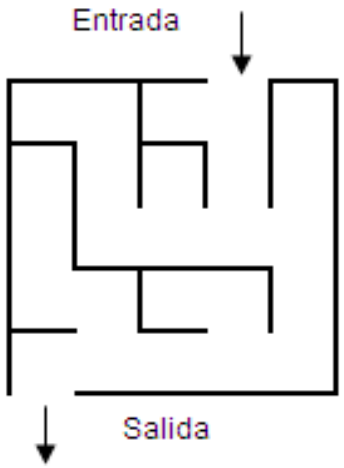

Figura 10. Maze perfecto

Existen varios algoritmos para la solución del Maze uno de ellos es el Algoritmo de la mano derecha o mano izquierda este algoritmo consiste en ir recorriendo el Maze tocando las paredes con la mano derecha (o izquierda). En este algoritmo el Robot tiene que ir recorriendo el Maze tocando la pared hasta encontrar la salida.

Este algoritmo es rápido, sin embargo este algoritmo no garantiza la ruta más corta. Figura 11.

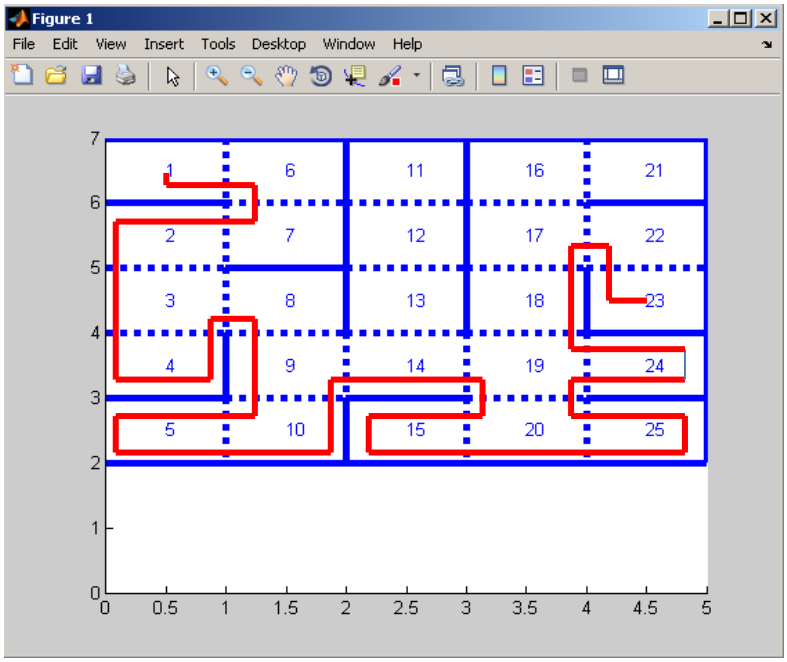

Figura 11. Recorrido utilizando el algoritmo de la mano derecha.

Este algoritmo solo funciona en Mazes simplemente conectados.

Capítulo 3

Metodología

3. METODOLOGIA

3.1 Localización Monte Carlo

Localización Monte Carlo (MCL) fue desarrollado por Dellaert y Fox [16] Ellos tomaron el método de filtrado de partículas utilizado en otras áreas como visión por computadora y aplicaron el método a la localización de robot móviles. Localización Monte Carlo es uno de los métodos más populares en la localización, debido a que ofrece soluciones a una gran variedad de problemas.

MCL es un filtro de Bayes recursivo que estima la distribución posterior de la posición del robot condicionado por el sensor de datos. Los Filtros de Bayes abordan el problema de estimar el estado *x* de un sistema dinámico (cadena de Markov parcialmente observable) a partir de mediciones del sensor. Los Filtros de Bayes asumen que el ambiente es de Markov, es decir, los datos pasado y futuro son (condicional) independientes si se conoce el estado actual.

La idea principal del filtro de Bayes es estimar una densidad de probabilidad sobre el estado espaciado condicionado sobre los datos. Este posterior que se suele llamar la *creencia (*Bel) y se denota

$$Bel(x_t) = p(x_t|d_{0\ldots t}) \qquad (5)$$

Aquí *x* denota el estado, *xt* es el estado en el tiempo *t* y *d0...t* denota el tiempo inicial desde *0* a *t*. Denotando *o* (observación) y *a* (acción), tenemos:

$$Bel(x_t) = p(x_t \mid o_t, a_{t-1},\ o_{t-1},\ a_{t-2}, \ldots,\ o_0) \qquad (6)$$

Sin la pérdida de la generalidad, suponemos que las observaciones y acciones llegan a una alternancia secuencial. Note que se debe utilizar, a_{t-1} para referirse a la lectura

odometrica que mide los movimientos que se produjo en el intervalo de tiempo [*t* -1; *t*], para ilustrar que el movimiento es el resultado de la acción de control en el tiempo *t* -1.

Los filtros de Bayes estiman la creencia de forma recursiva. La creencia inicial caracteriza el conocimiento inicial sobre el estado del sistema. En ausencia de tal conocimiento, suele ser inicializado por una distribución uniforme en el espacio de estados. Para derivar una ecuación recursiva de actualización, se observa que la expresión (6) se puede transformar por la regla de Bayes de la siguiente forma:

$$Bel(x_t) = \frac{p(o_t|\, x_t, a_{t-1}, \ldots, o_0)\, p(x_t\, |a_{t-1}, \ldots, o_0)}{p(o_t\, |a_{t-1}, \ldots, o_0)} \qquad (7)$$

Debido a que el denominador es una constante respecto a la variable x_t, la regla de Bayes es por lo general escrita como:

$$Bel(x_t) = \eta p(o_t|\, x_t, a_{t-1}, \ldots, o_0) p(x_t | a_{t-1}, \ldots, o_0) \qquad (8)$$

Donde η es la constante de normalización

$$\eta = p(o_t | a_{t-1}, \ldots, o_0)^{-1} \qquad (9)$$

Como se ha observado anteriormente, los filtros de Bayes descansan en el supuesto de que los datos futuros son independientes de los datos anteriores teniendo el conocimiento del estado actual. Una suposición típicamente es denominado la suposición de Markov . La suposición de Markov implica

$$p(o_t|\, x_t, a_{t-1}, \ldots, o_0) = p(o_t|\, x_t) \qquad (10)$$

y por lo tanto el objetivo expresión (7) se puede simplificar a:

$$Bel(x_t) = \eta p(o_t|\, x_t) p(x_t|a_{t-1}, \dots, o_0) \qquad (11)$$

De esta forma, al integrar más a la derecha sobre el estado en el tiempo *t* -1:

$$Bel(x_t) = \eta p(o_t|\, x_t) \int p(x_t|x_{t-1}, a_{t-1}, \dots, o_0)\ p(x_{t-1}|a_{t-1}, \dots, o_0) dx_{t-1} \qquad (12)$$

Una vez más, se puede aprovechar el supuesto de Markov para simplificar $p(x_t|x_{t-1}, a_{t-1}, \dots, o_0)$:

$$p(x_t|x_{t-1}, a_{t-1}, \dots, o_0) = p(x_t|x_{t-1}, a_{t-1}) \qquad (13)$$

Se obtiene la siguiente expresión:

$$Bel(x_t) = \eta p(o_t|\, x_t) \int p(x_t|x_{t-1}, a_{t-1})\ p(x_{t-1}|a_{t-1}, \dots, o_0) dx_{t-1} \qquad (14)$$

Sustituyendo la definición básica de la creencia de nuevo en *Bel* (14), se obtiene la importante ecuación recursiva

$$Bel(x_t) = \eta p(o_t|\, x_t) \int p(x_t|x_{t-1}, a_{t-1})\, Bel(x_{t-1}) dx_{t-1} \qquad (15)$$

Esta ecuación es la ecuación recursiva de actualización por filtros de Bayes.

Para poner en práctica la ecuación (15), se necesita conocer dos densidades condicionales: la probabilidad $p(x_t|x_{t-1}, a_{t-1})$, que se refiere a la densidad del

siguiente estado o el modelo de movimiento, y la densidad $p(o_t|x_t)$, que comúnmente se denomina modelo de percepción o el modelo del sensor. Ambos modelos suelen ser fijos (también llamado: invariantes en el tiempo), es decir, no dependen del tiempo *t*. Esta estacionalidad nos permite simplificar la notación que denota por estos modelos *p*(*x'*|*x*,*a*), y *p*(*o*|*x*), respectivamente.

El modelo de movimiento, *p*(*x'*|*x,a*), es una generalización probabilística de la cinemática del robot. Para un robot que operan en el plano las posiciones de x y x' son variables en tres dimensiones.

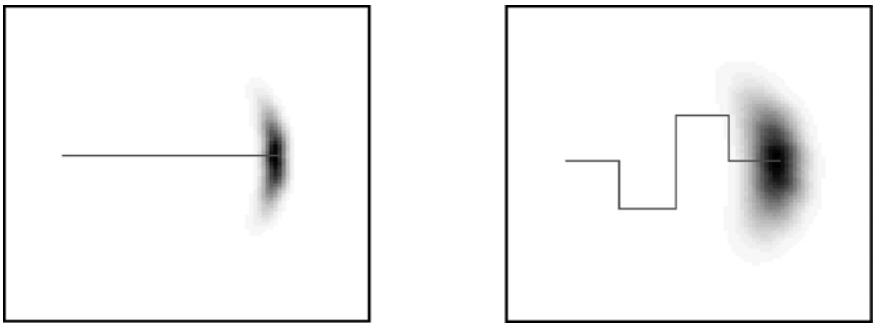

Figura 12. Densidad *p(x'|x,a)* después de mover 40 metros (diagrama izquierdo) y 80 metros (diagrama de la derecha). [17]

La Figura 12. Muestra dos ejemplos de *p(x'|x,a)*. En ambos ejemplos, la postura inicial de x se muestra a la izquierda, y la línea continua representa los datos de odometría, medido por el robot. El área sombreada de la derecha muestra la densidad *p(x'|x,a),* la posición más oscura, es la que tiene mayor probabilidad. Una comparación de los dos diagramas revela que el margen de incertidumbre depende del movimiento global.

A pesar de que la postura de un robot libre de ruido es la misma para ambos segmentos de movilidad, la incertidumbre en el diagrama de la derecha es más grande debido a la mayor distancia recorrida por el robot.

Un modelo de toma de muestras es una rutina que acepta *x* y *a* como entrada y genera al azar posiciones x' distribuidas de acuerdo a$p(x'|x,a)$. La Figura 13 muestra un ejemplo de un modelo $p(x'|x,a)$, aplicado a una secuencia de medidas de odometría, como lo indica la línea. La secuencia de conjuntos de partículas se aproxima a la densidad de un robot que mide sólo odometría.

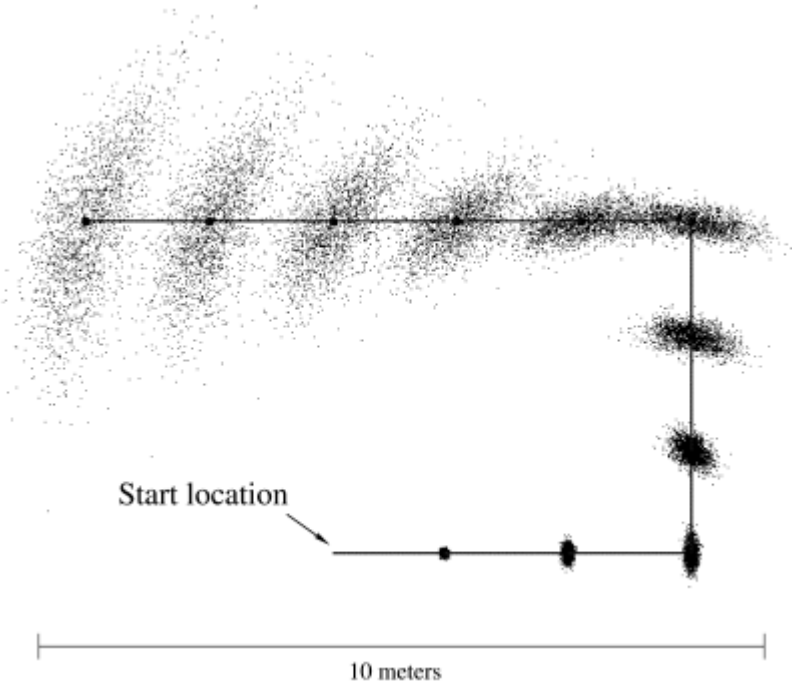

Figura 13. Aproximación de muestreo basado en Odometría[17].

En la figura 13 se muestra las acciones (línea sólida), y las muestras representan las creencias del robot en diferentes puntos en el tiempo. [17]

Para el caso del modelo perceptivo $p(o|x)$,los robots móviles son normalmente equipados con sensores de rango, tales como sensores de sonido o sensores láser de largo alcance.

La idea de MCL es representar la creencia Bel(*x*) por un conjunto de muestras de pesos *m* distribuidos de acuerdo a Bel(*x*) (ecuación 16):

$$Bel(x_t) \approx \left\{x^{(i)}, w^{(i)}\right\}_{i=1,\ldots,m} \qquad (16)$$

Cada $x^{(i)}$ es una muestra de la variable aleatoria x , es decir, la hipótesis de un estado (pose). El $w^{(i)}$ numérico no negativo parámetros se llaman factores de importancia, tal que la suma sea 1 (aunque esto no es estrictamente necesario ende partículas, filtrado). Como su nombre indica, los factores de importancia determinar el peso (Importancia) de cada muestra. El conjunto de muestras, por lo tanto, define una función de probabilidad discreta que se aproxima a la continua creencia Bel (*x*).

El conjunto inicial de muestras representa el conocimiento inicial de *Bel* (x_0) sobre el estado de la sistema dinámico. Por ejemplo, en localización global robot móvil, la creencia inicial es una serie de poses elaborado de acuerdo a una distribución uniforme en el universo robot, anotado por el factor de importancia uniforme de 1/*m*. Si la postura inicial es conocido hasta cierto margen pequeño de error, *Bel*(x_0) puede ser inicializado por las muestras tomadas desde una estrecha gaussiana centrada a la posición correcta. La actualización recursiva se realiza en tres pasos:

(1) La muestra $x^{(i)}_{t-1}$~ *Bel*(x_{t-1}) de los pesos que es representada por Bel(x_{t-1}) . Cada partícula tales $x^{(i)}_{t-1}$ se distribuyen de acuerdo a la creencia de distribución Bel(x_{t-1}).

(2) La muestra $x^{(i)}$~ p(x_t| $x^{(i)}_{t-1}$, a_{t-1}) . La pareja$\langle x^{(i)}_t, x^{(i)}_{t-1} \rangle$ es distribuido de acuerdo con la distribución del producto

$$q_t := p\,(x \mid x_{t-1}, a_{t-1}) \times Bel\,(x_{t-1}). \qquad (17)$$

(3) Por último, para corregir la asimetría entre la distribución q_t propuesta y la distribución deseada que se especifica en la ecuación. (16) y volviendo a mencionar la muestra $\langle x^{(i)}_{t-1}, x^{(i)}_t \rangle$:

$$\eta p(o_t \mid x_t^{(i)}) p\,(x \mid x_{t-1}, a_{t-1})\, Bel\,(x^{(i)}_{t-1}) \qquad (18)$$

Esta falta de coincidencia se debe al factor de importancia (no normalizada), que se asigna a la *i* esima muestra:

$$w^{(i)} = p(o_t \mid x_t^{(i)}). \qquad (19)$$

El factor de importancia se obtiene del cociente de la distribución objetivo (18) y la distribución propuesta (19), hasta un factor de escala constante:

$$\frac{\eta p(o_t \mid x^{(i)}_t) p(x_t \mid x_{t-1}) Bel(x^{(i)}_{t-1})}{p(x_t \mid x_{t-1}, a_{t-1}) Bel(x^{(i)}_{t-1})} = \eta p(o_t \mid x^{(i)}_t) \qquad (20)$$

El cociente es proporcional a $w^{(i)}$, ya que η es una constante.

La rutina de muestreo se repite *m* veces, produciendo un conjunto de *m* muestras $x_t^{(i)}$. (Con $i = 1,..., m$). Posteriormente, los factores de importancia son normalizados de manera que la suma sea 1 y por lo tanto define una distribución de probabilidad discreta.

El pseudocódigo del algoritmo MCL se expresa de la siguiente manera:

Algoritmo MCL(X, a, o)

$X' = \varnothing$

for i = 0 to m do

generar al azar x desde X de acuerdo a $w_1, \ldots, w_m$

generar al azar $x' \sim p(x' \mid a, x)$

$w' = p(o \mid x')$

add (x', w') to X

endfor

Normalizar el factor de importancia w' en X'

Regresar X'

3.2 Simulación de la trayectoria de un Robot Móvil

Para simular la trayectoria de un Robot se desarrolló una aplicación en Matlab.

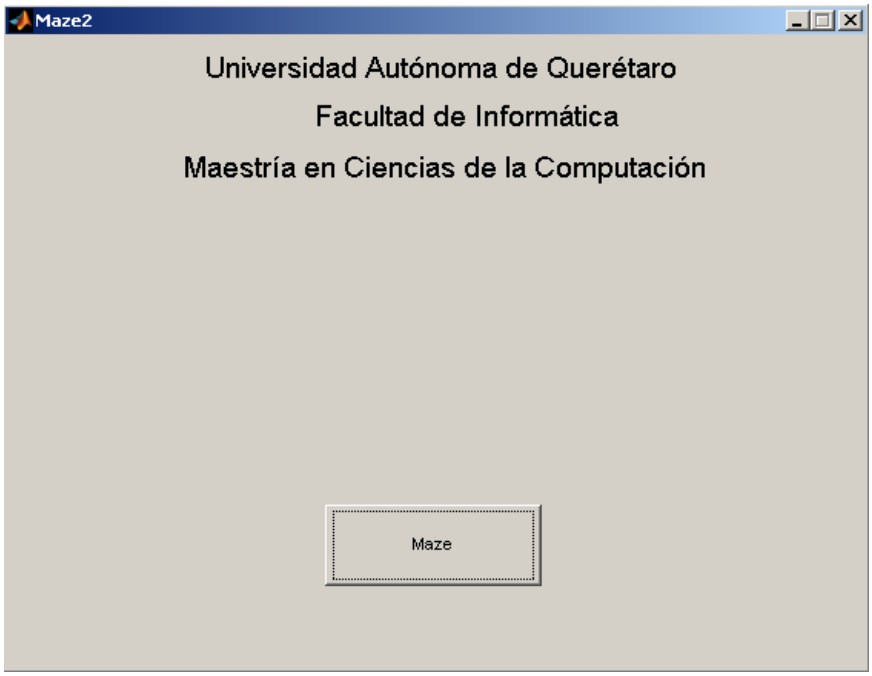

Figura 14. Pantalla de inicio de la aplicación en Matlab.

Primero se construye el ambiente por el cual Robot tendrá que navegar. El ambiente por donde el robot tendrá que navegar es un Maze (en este caso de tamaño 5 x 5), como se muestra en la Figura 15. El Robot iniciara en el nodo 1 hasta alcanzar el nodo su meta, el nodo 25.

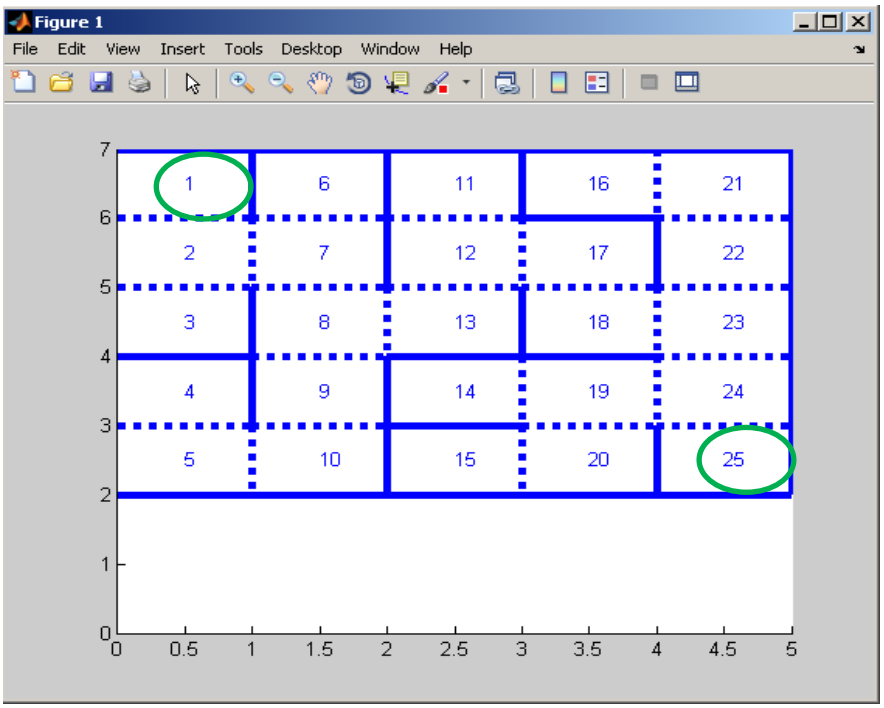

Figura 15. Generación del Maze.

En la Figura 15. Las líneas solidas indican que existe una pared y las líneas punteadas son puertas por donde el Robot puede navegar hasta encontrar su objetivo. Para encontrar el camino más corto dentro del maze, se utilizara el algoritmo de Dijkstra y la teoría de grafos. Primero, se construye un grafo dirigido en donde cada arista tendrá un peso de 1 y después se aplica el algoritmo de Dijkstra, como se muestra en la Figura 16.

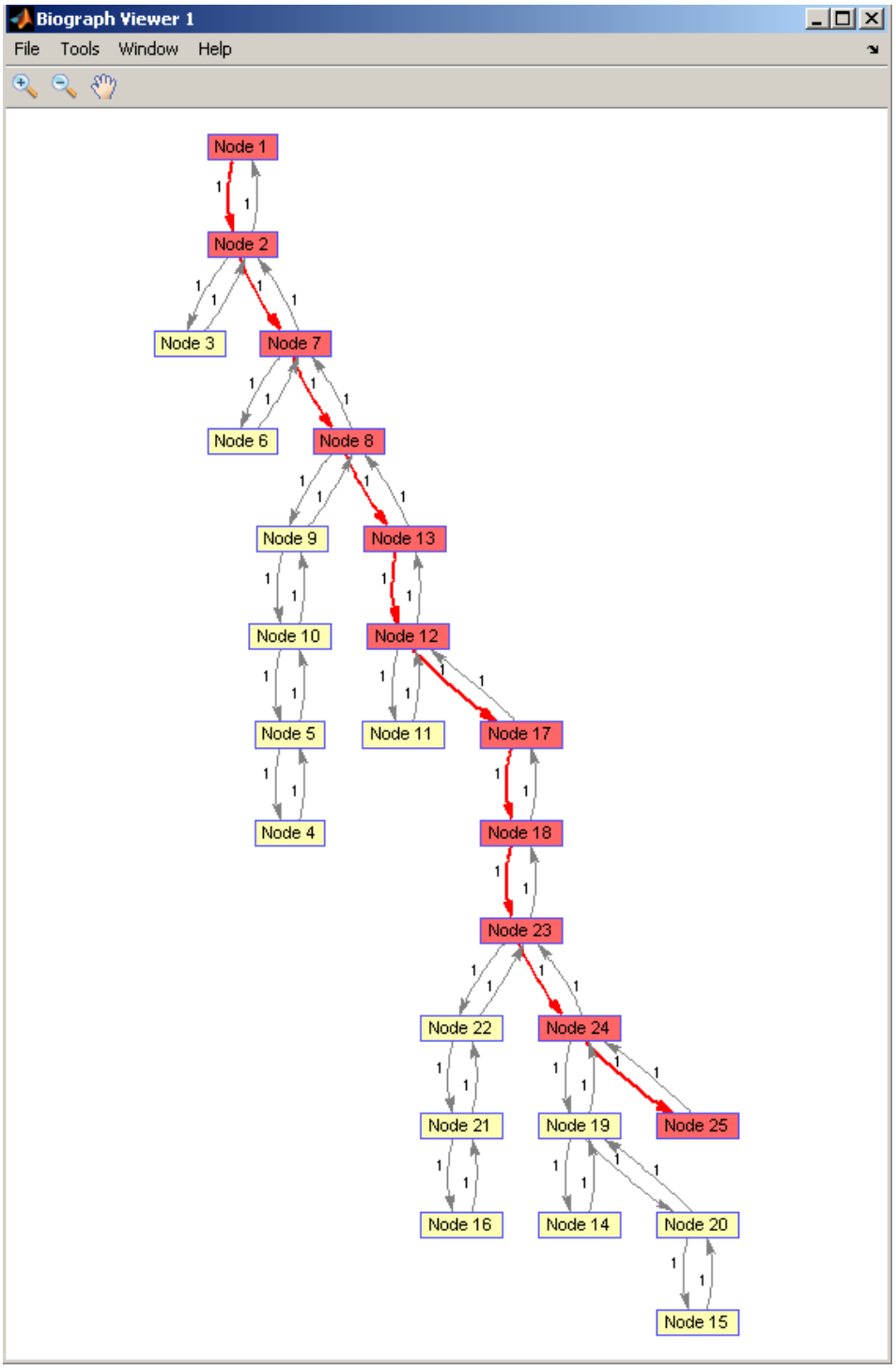

Figura 16. Generación de nodos del Maze y obtención de la ruta más corta.

En el ejemplo de las figuras 15 y 16 se puede observar que la ruta más corta es; 1,2,7,8,13,12,17,18,23,24,25. Ya que se tiene la ruta más corta, se traza la trayectoria ideal por la cual navegara el Robot.

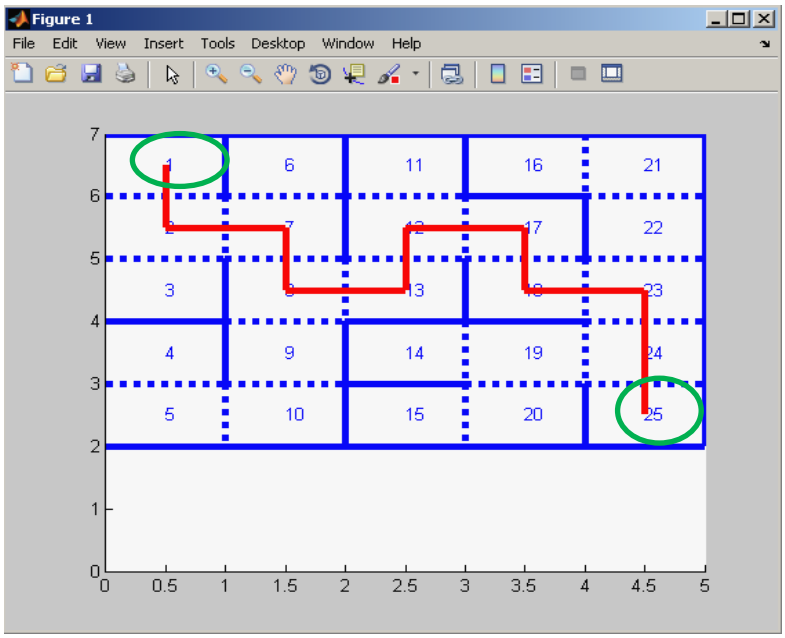

Figura 17. Ruta más corta.

Se obtiene el recorrido del Robot utilizando el método Monte Carlo. Con el cual se realiza la simulación de la trayectoria del Robot Móvil.

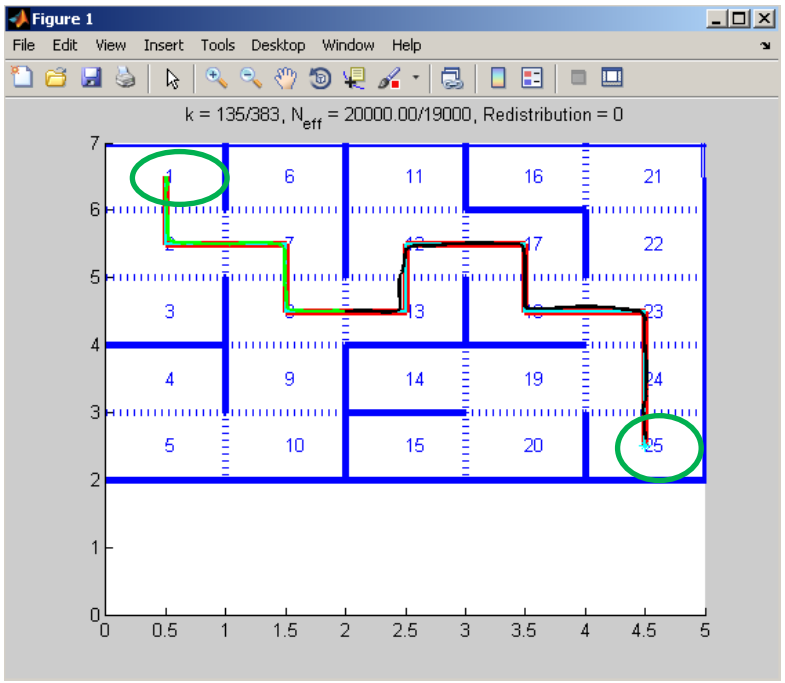

Figura 18. Simulación de la navegación del Robot en una ruta establecida. Fin del recorrido del Robot.

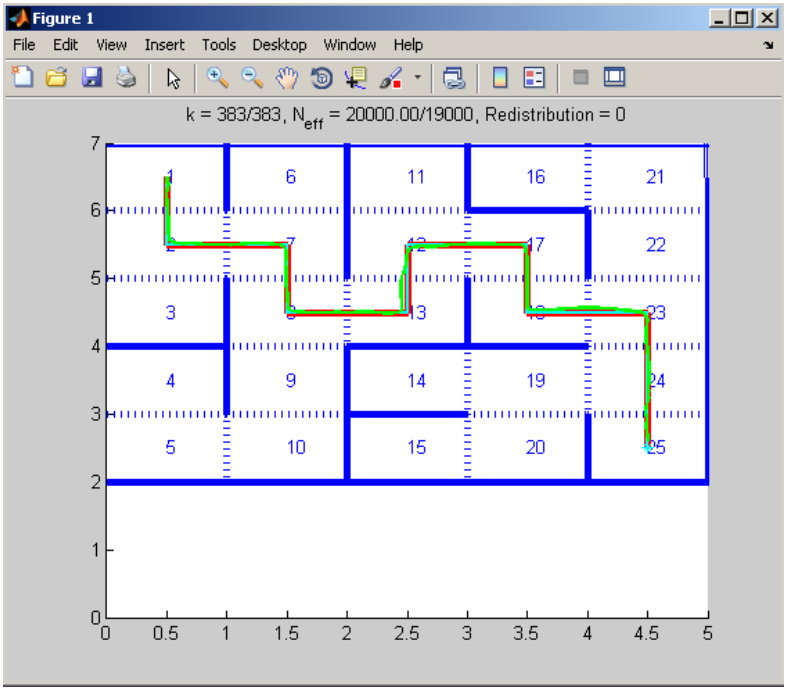

Figura 19. Fin de la navegación del Robot en una ruta establecida.

A continuación se obtendrán varios experimentos con un Maze de tamaño 5 * 5. En cada experimento el Maze será diferente y en cada experimento el nodo inicial y el nodo final será seleccionado al azar, para mostrar la efectividad de dicho algoritmo.

Experimento 1.

La Figura 20.Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **2** y la meta es el nodo **23**.

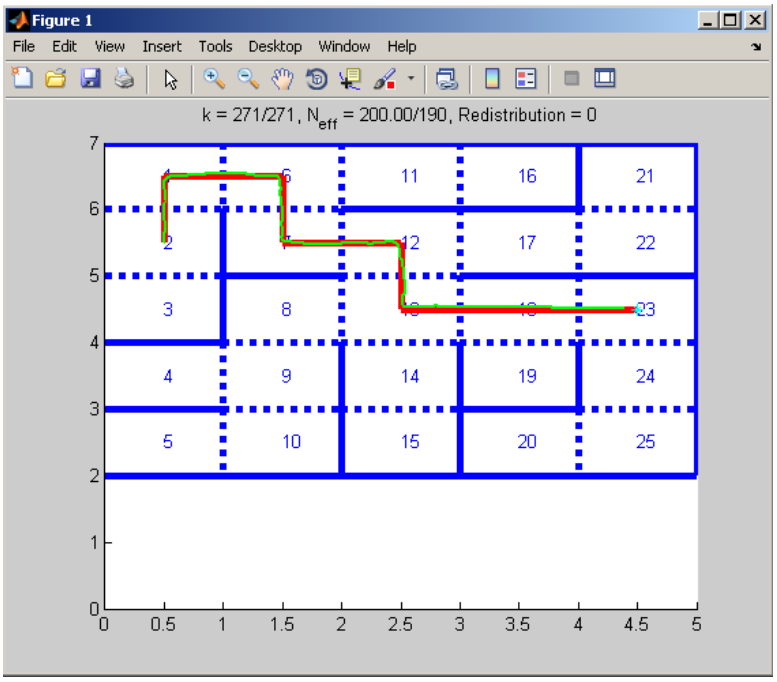

Figura 20. Maze y recorrido del robot. Experimento 1, maze 5 * 5.

La Figura 21 muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **2** al nodo final **23**.

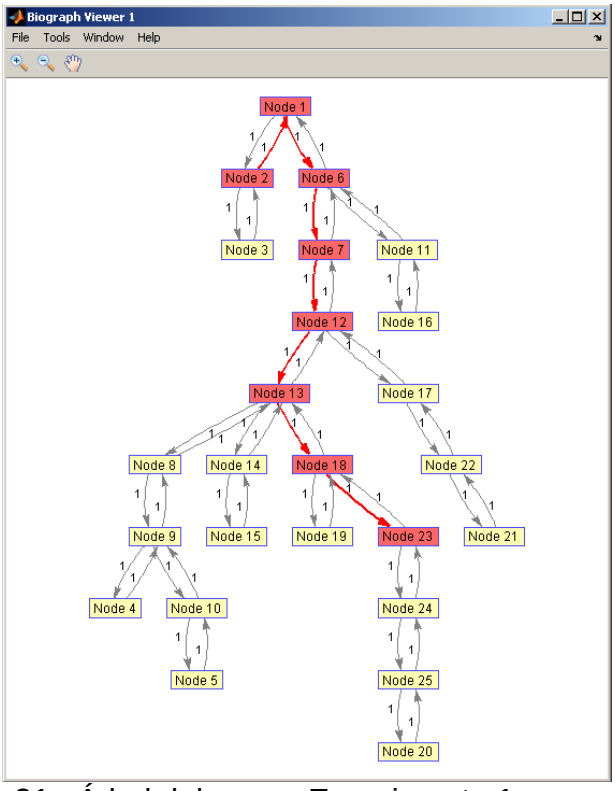

Figura 21. Árbol del maze. Experimento 1, maze 5 * 5.

Experimento 2.

La Figura 22.Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **1** y la meta es el nodo **24**.

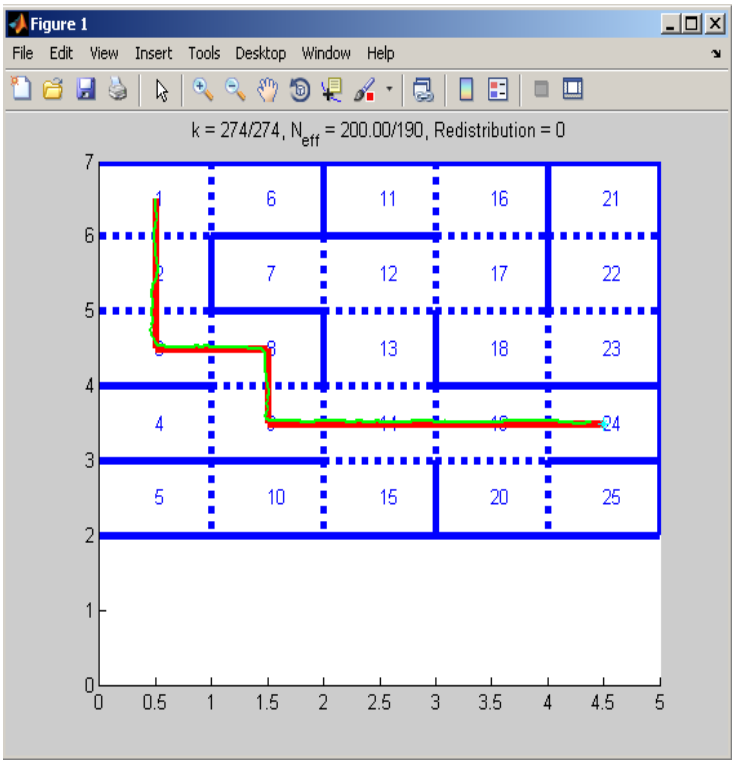

Figura 22. Maze y recorrido del robot. Experimento 2, maze 5 * 5.

La Figura 23. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **1** al nodo final **24**.

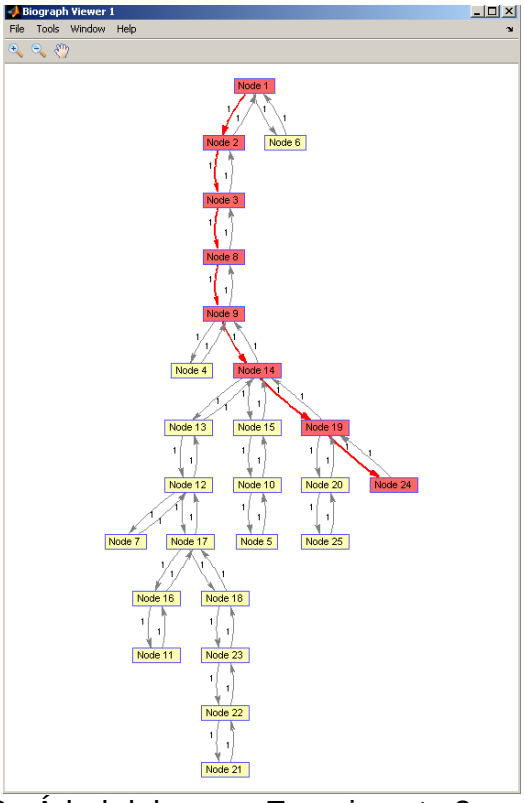

Figura 23. Árbol del maze. Experimento 2, maze 5 * 5.

Experimento 3.

La Figura 24. Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **4** y la meta es el nodo **22**.

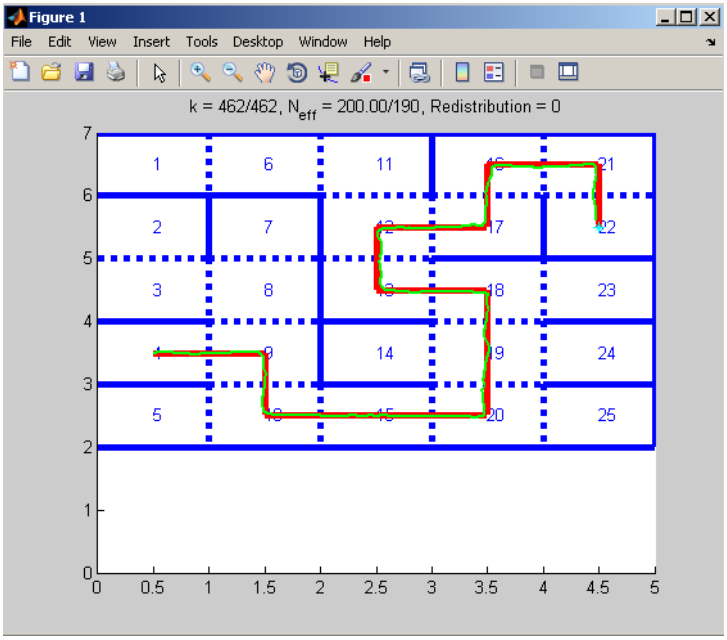

Figura 24. Maze y recorrido del robot. Experimento 3, maze 5 * 5.

La Figura 25. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **4** al nodo final **22**.

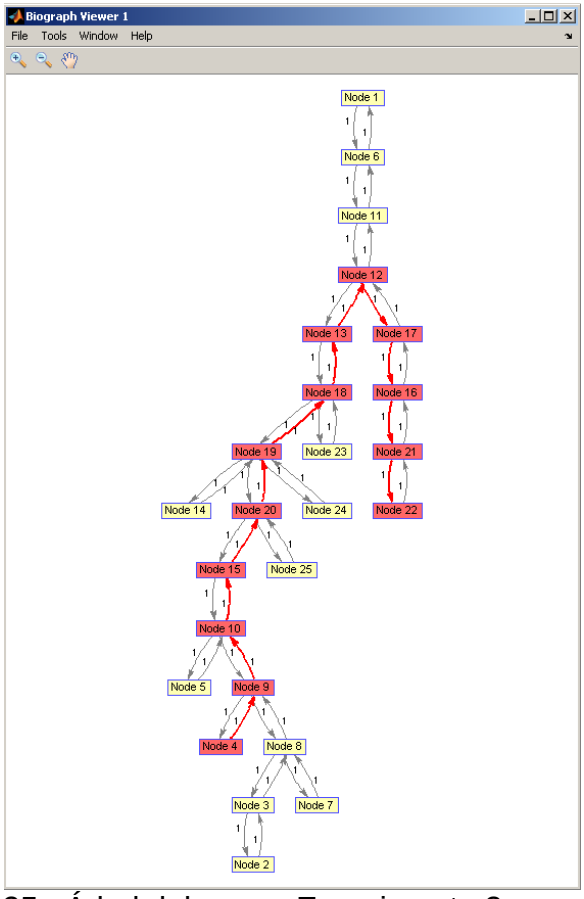

Figura 25. Árbol del maze. Experimento 3, maze 5 * 5.

Experimento 4.

La Figura 26. Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **4** y la meta es el nodo **22**.

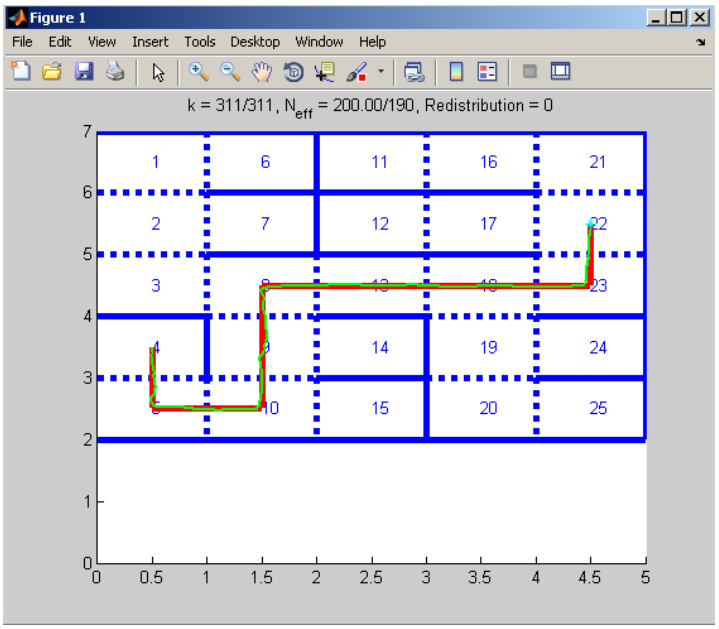

Figura 26. Maze y recorrido del robot. Experimento 4, maze 5 * 5.

La Figura 27.Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **4** al nodo final **22**.

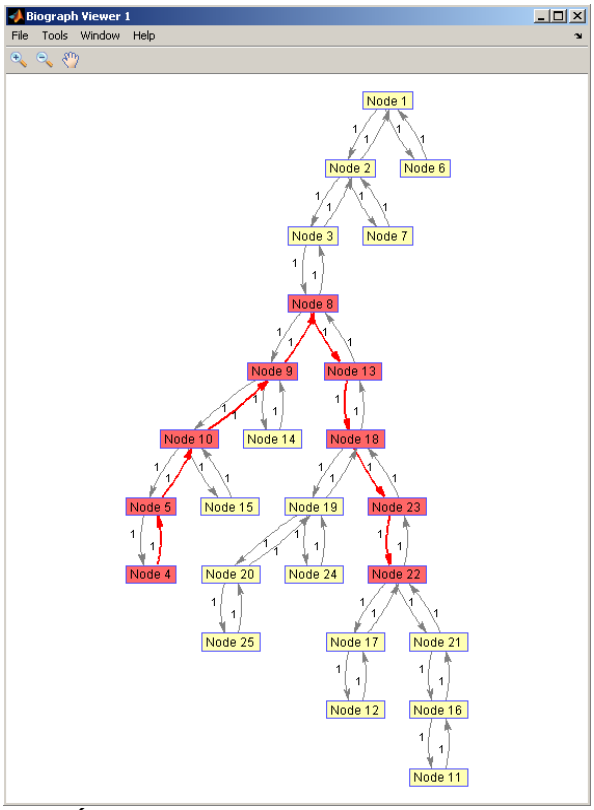

Figura 27. Árbol del maze. Experimento 4, maze 5 * 5.

Experimento 5.

La Figura 28.Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **4** y la meta es el nodo **24**.

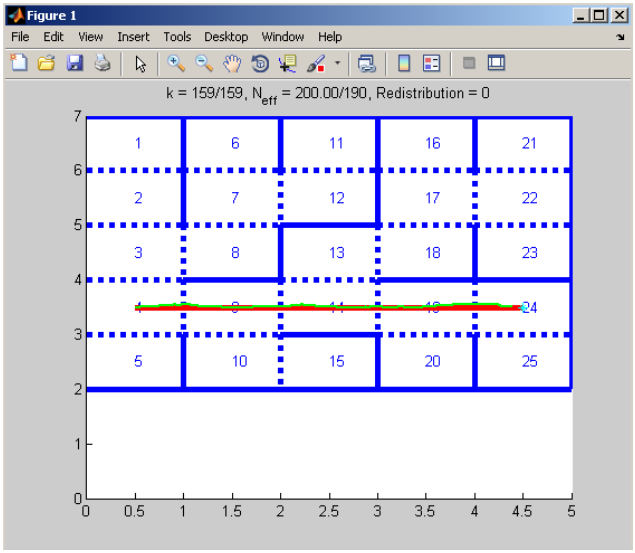

Figura 28. Maze y recorrido del robot. Experimento 5, maze 5 * 5.

La Figura 29.Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **4** al nodo final **24**.

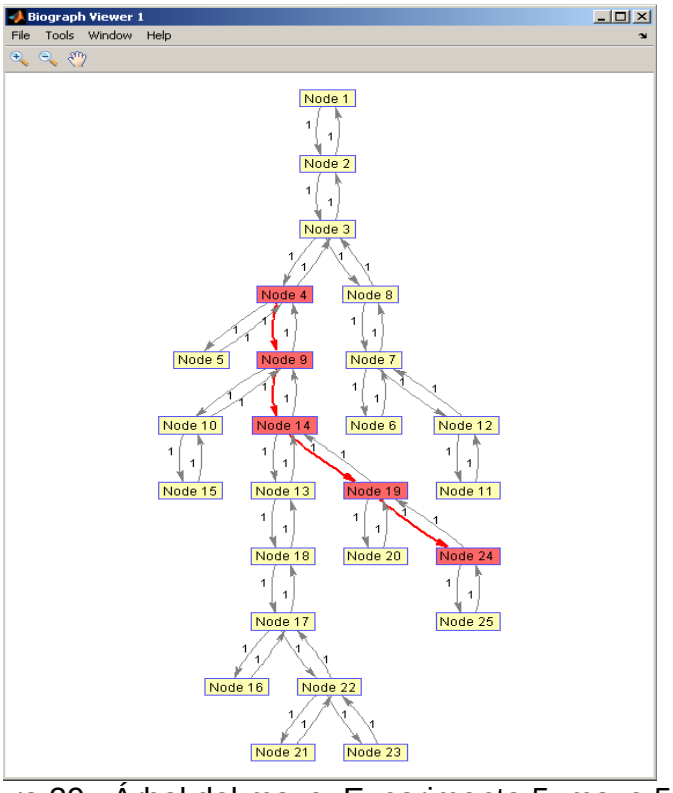

Figura 29. Árbol del maze. Experimento 5, maze 5 * 5.

Experimento 6.

La Figura 30. Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **1** y la meta es el nodo **23**.

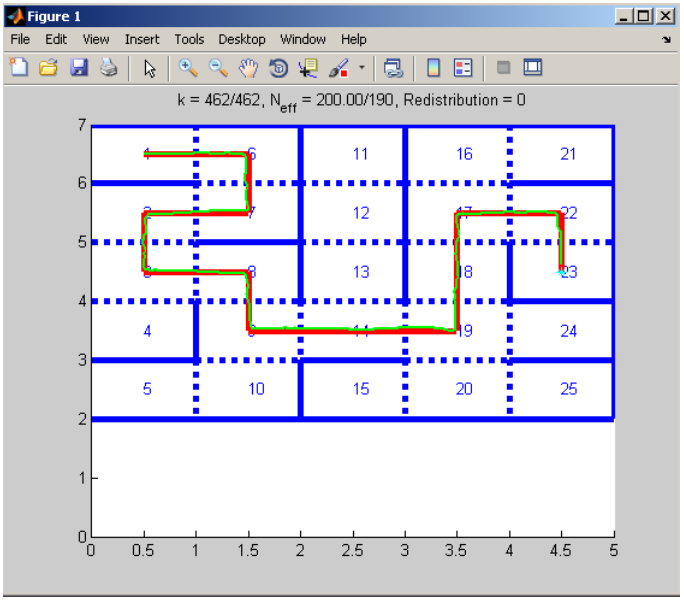

Figura 30. Maze y recorrido del robot. Experimento 6, maze 5 * 5.

La Figura 31. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **1** al nodo final **23**.

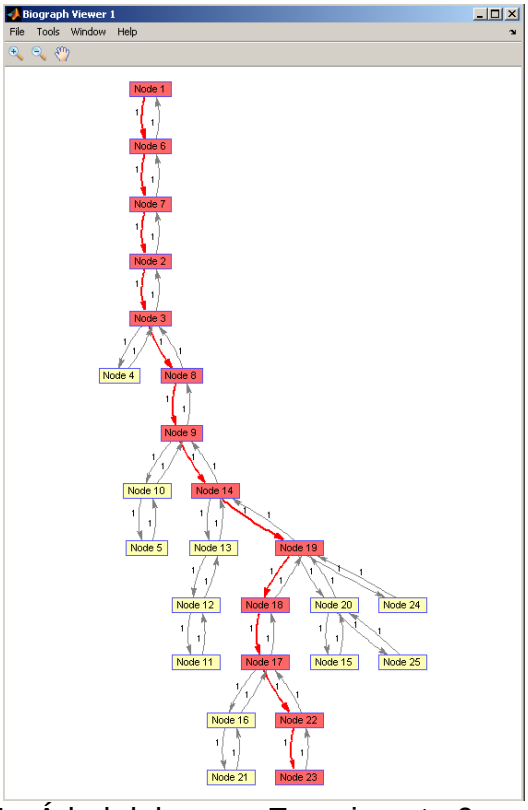

Figura 31. Árbol del maze. Experimento 6, maze 5 * 5.

Experimento 7.

La Figura 32. Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **1** y la meta es el nodo **22**.

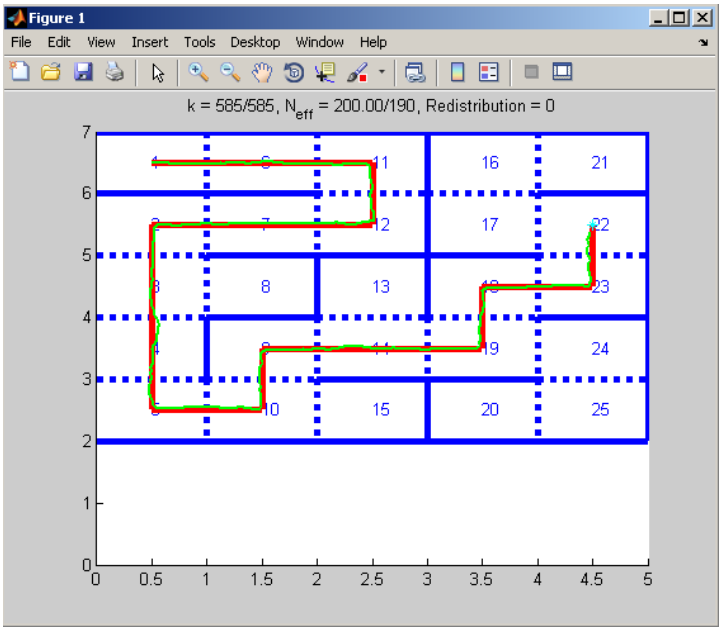

Figura 32. Maze y recorrido del robot. Experimento 7, maze 5 * 5.

La Figura 33. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **1** al nodo final **22**.

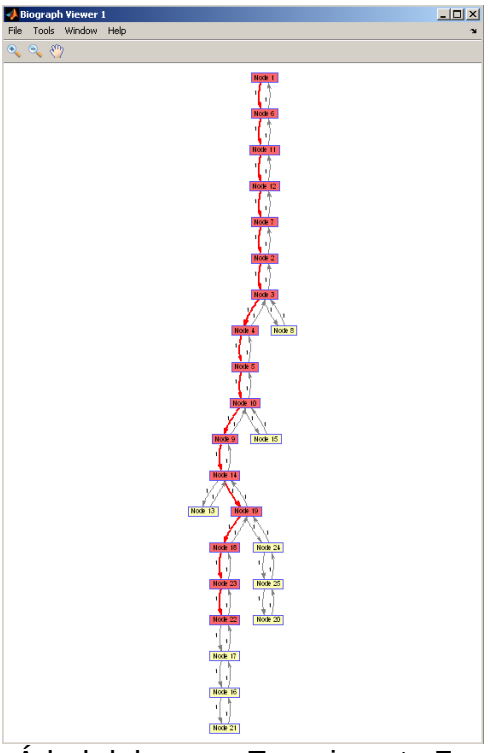

Figura 33. Árbol del maze. Experimento 7, maze 5 * 5.

Experimento 8.

La Figura 34. Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **4** y la meta es el nodo **23**.

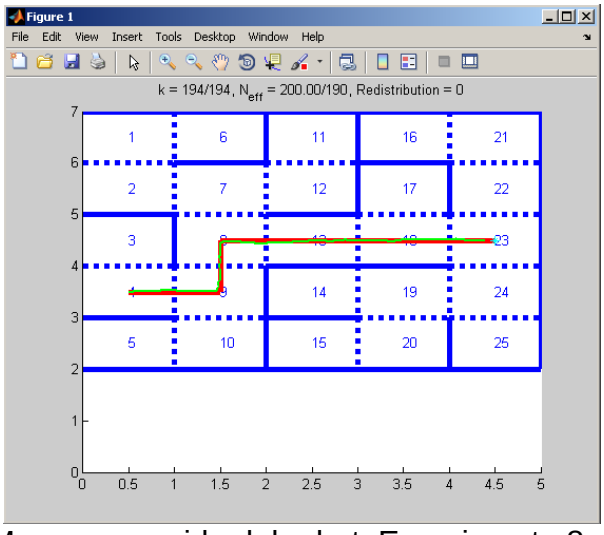

Figura 34. Maze y recorrido del robot. Experimento 8, maze 5 * 5.

La Figura 35. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **4** al nodo final **23**.

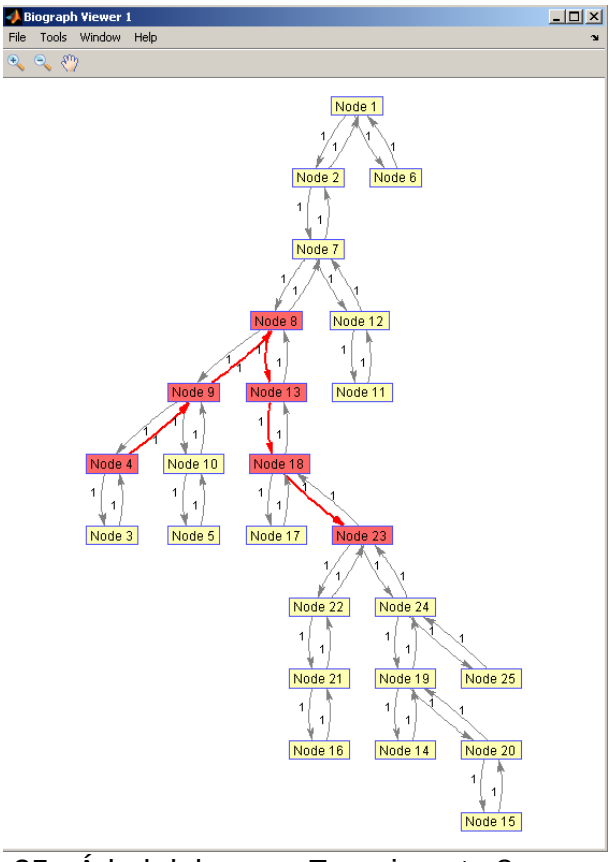

Figura 35. Árbol del maze. Experimento 8, maze 5 * 5.

Experimento 9.

La Figura 36. Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **3** y la meta es el nodo **23**.

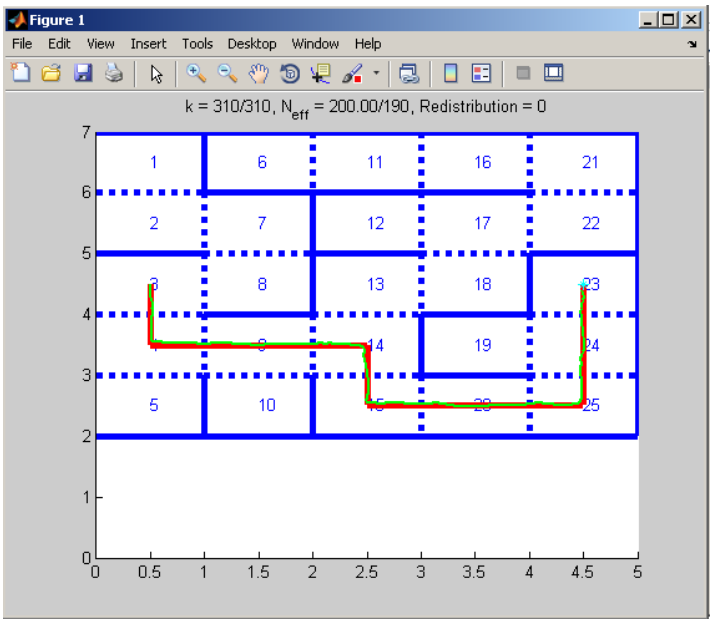

Figura 36. Maze y recorrido del robot. Experimento 9, maze 5 * 5.

La Figura 37. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **3** al nodo final **23**.

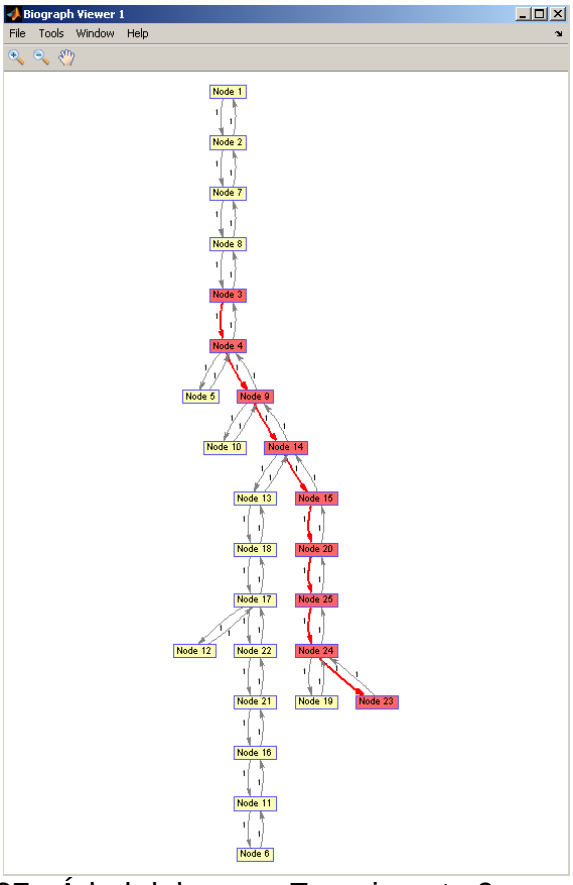

Figura 37. Árbol del maze. Experimento 9, maze 5 * 5.

Experimento 10.

La Figura 38. Muestra un Maze 5 * 5 en donde el recorrido inicia en el nodo **5** y la meta es el nodo **22**.

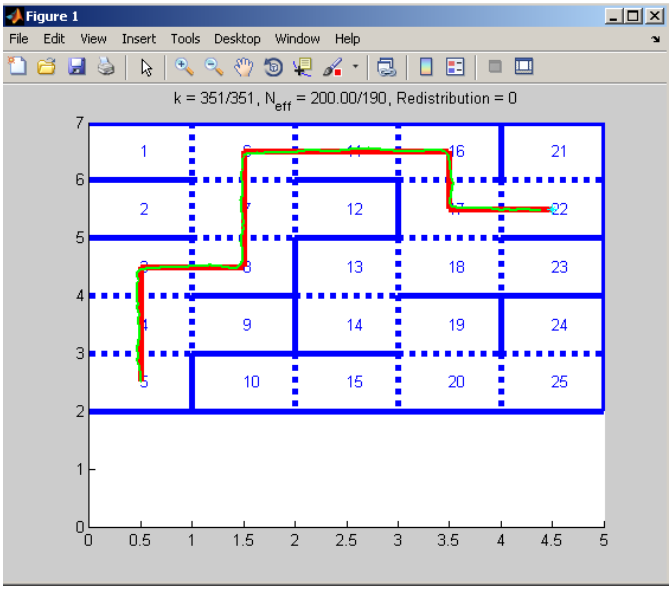

Figura 38. Maze y recorrido del robot. Experimento 10, maze 5 * 5.

La Figura 39. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **5** al nodo final **22**.

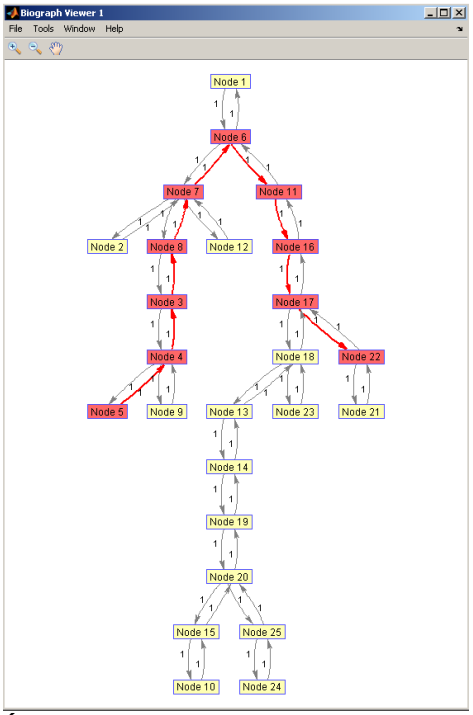

Figura 39. Árbol del maze. Experimento 10, maze 5 * 5.

A continuación se muestran los tiempos obtenidos de cada experimento realizado con un maze de 5 * 5. Los tiempos se obtuvieron utilizando la función *tic, toc* de Matlab. El número de partículas utilizadas para el algoritmo de MCL fue de 200.

Numero de Experimento	Tiempo
1	8.14E-04
2	9.62E-04
3	3.96E-04
4	4.68E-04
5	7.73E-04
6	1.16E-01
7	2.51E-04
8	2.47E-04
9	2.50E-04
10	2.52E-04

Tabla 1. Tiempos para obtener la ruta más corta maze 5 * 5.

Numero de Experimento	Tiempo
1	2.94E+01
2	3.07E+01
3	9.04E+01
4	4.20E+01
5	1.22E+01
6	8.84E+01
7	1.52E+02
8	2.47E-04
9	3.56E+01
10	4.88E+01

Tabla 2. Tiempos para recorrer la trayectoria MCL maze 5 * 5.

En la Tabla 2.Se muestran los tiempos en que el robot tarda en recorrer la trayectoria previamente obtenida utilizando el algoritmo de MCL. El algoritmo llena todas las partículas en todo el maze y de acuerdo a su peso va filtrando las partículas y otro factor importante es que cada que avanza manda la información al robot y la vuelve a procesar.

MCL revisa el Maze completo con sus salidas y detecta la ruta más óptima ya predefinida dentro del robot. Para esta ruta ya predefina utilizamos dijkstra y solamente utilizamos MCL para que el robot siga la trayectoria. Ya que si utilizamos MCL para la optimización sería más tardado y difícil de implementar.

En la Tabla 2. Se muestra que el **Experimento 5** con un tiempo de **1.22E+01** es el mejor caso. Esto se debe a que el robot solo pasa por 5 nodos y tiene que navegar en línea recta. En la Tabla 2. También se muestra el peor caso y es el **Experimento 3** con un tiempo de **9.04E+01**. En este experimento se observa que tiene que pasar por 13 nodos y su trayectoria no es en línea recta lo que hace que recorrer una mayor cantidad de nodos.

Para comprobar el comportamiento de los algoritmos se realizaron los siguientes experimentos. Se realizó con un Maze de 2 * 2 en donde el nodo inicial es **2** y el nodo final es el **3**. Figura 40.

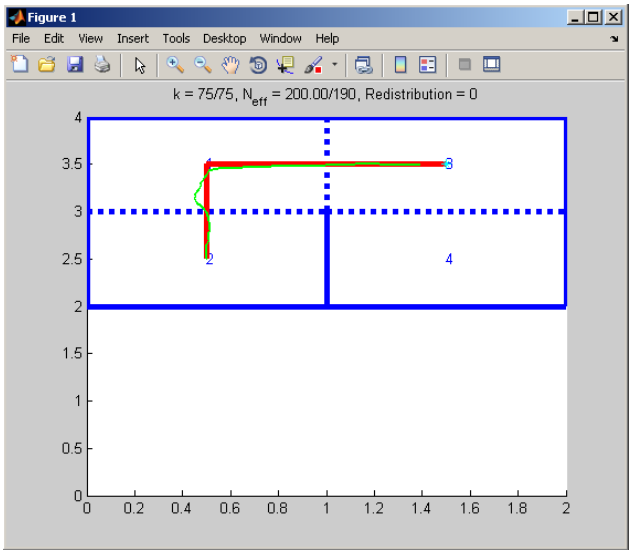

Figura 40. Maze y recorrido del robot. Maze 2 * 2.

La Figura 41.Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **2** al nodo final **3**.

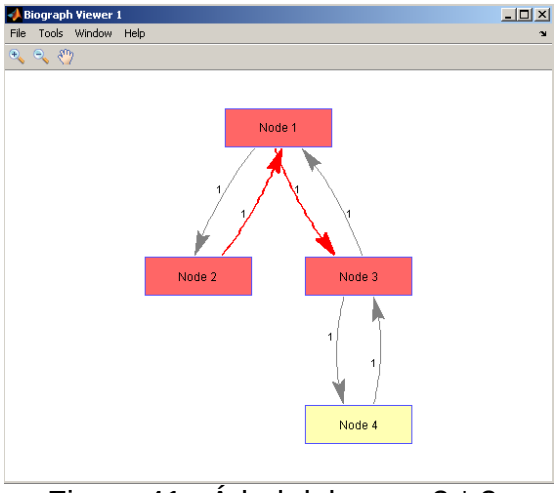

Figura 41. Árbol del maze 2 * 2.

El tiempo obtenido en la ruta más corta fue 0.17073399 y el tipo obtenido en MCL fue 3.5898266.

Otro experimento se realizó con un Maze de 6 * 6 en donde el nodo inicial es **4** y el nodo final es el **16**. Figura 42.

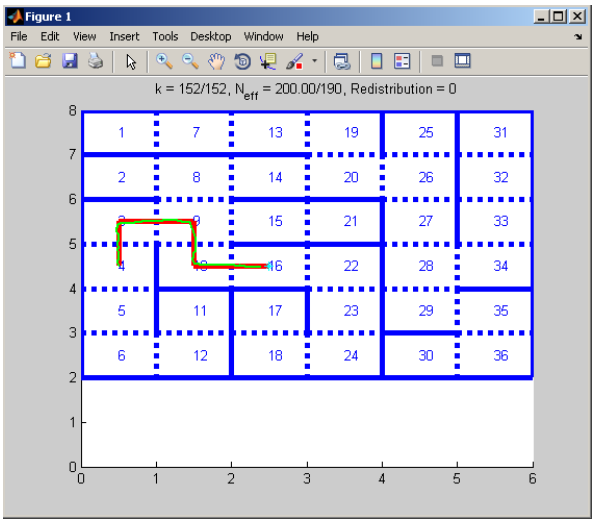

Figura 42. Maze y recorrido del robot. Maze 6 * 6, nodo final 16.

La Figura 43. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **4** al nodo final **16**.

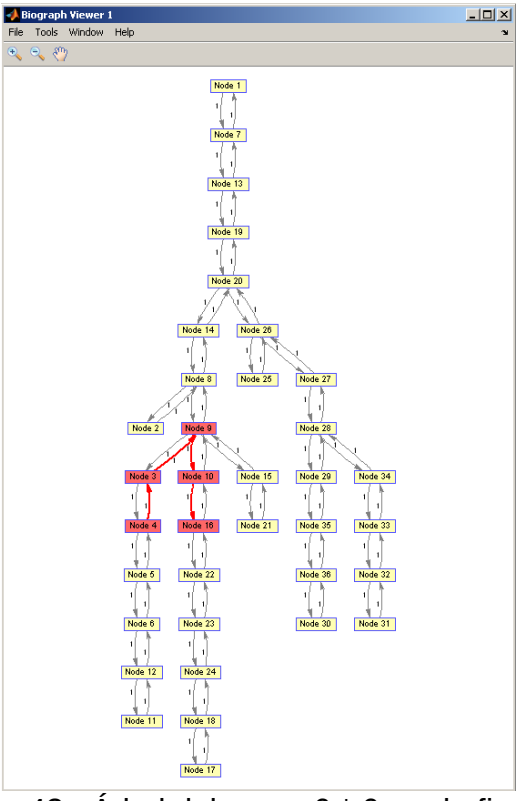

Figura 43. Árbol del maze 6 * 6, nodo final 16.

El tiempo obtenido en la ruta más corta fue 0.00027908575 y el tipo obtenido en MCL fue 11.24741.

Se realizó un experimento con un Maze de 6 * 6 en donde el nodo inicial es **6** y el nodo final es el **32**. Figura 44.

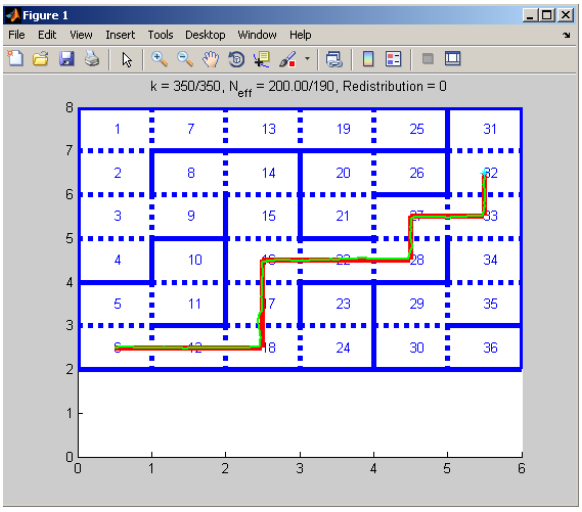

Figura 44. Maze y recorrido del robot. Maze 6 * 6, nodo final 32.

La Figura 45. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **6** al nodo final **32**.

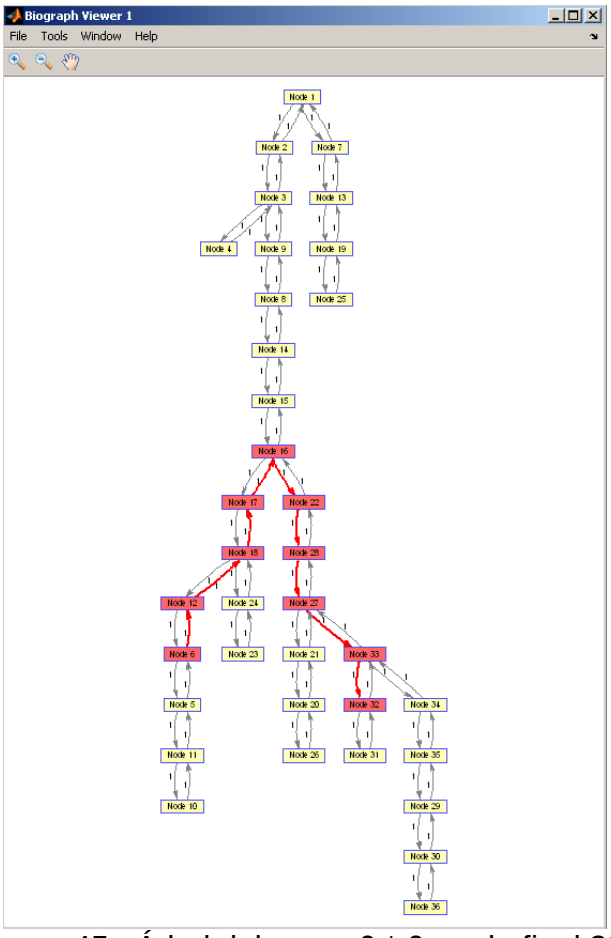

Figura 45. Árbol del maze 6 * 6, nodo final 32.

El tiempo obtenido en la ruta más corta fue 0.00027293972 y el tipo obtenido en MCL fue 47.352557.

Otro experimento se realizó con un Maze de 7 * 7 en donde el nodo inicial es **2** y el nodo final es el **48**. Figura 46

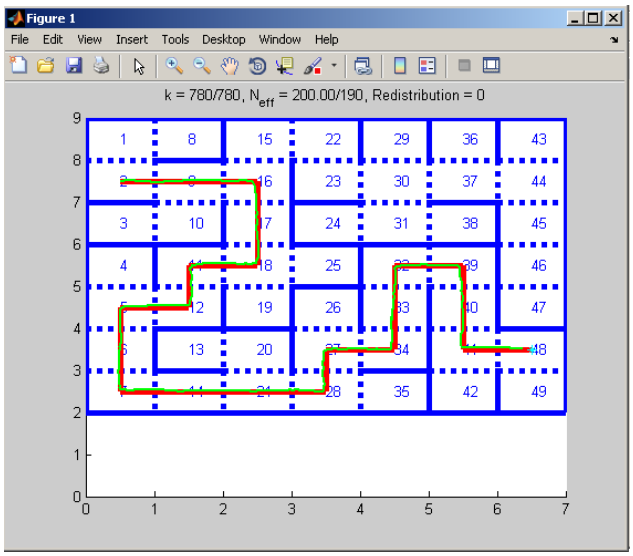

Figura 46. Maze y recorrido del robot. Maze 7 * 7.

La Figura 47. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **6** al nodo final **48**.

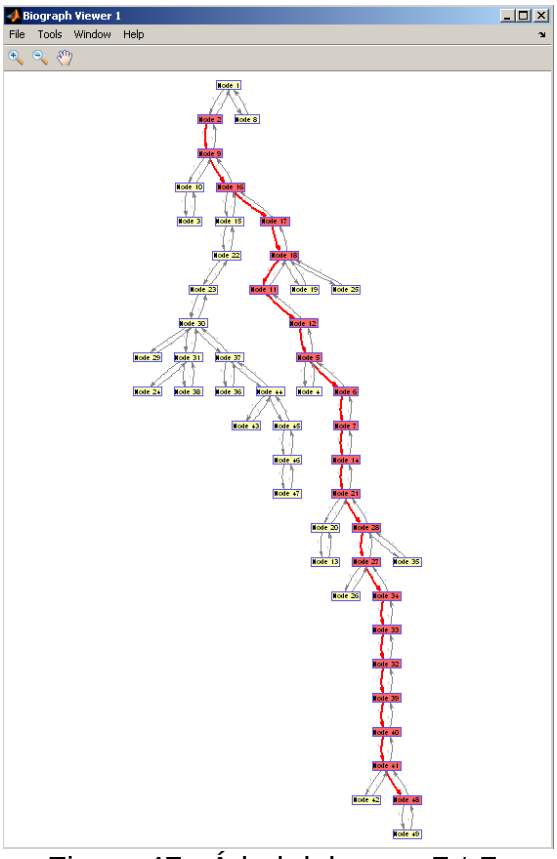

Figura 47. Árbol del maze 7 * 7.

El tiempo obtenido en la ruta más corta fue 0.00032099052 y el tipo obtenido en MCL fue 286.20395

A continuación se muestran los tiempos obtenidos con un maze de 7 * 7.

Numero de Experimento	Tiempo
1	3.97E-04
2	3.28E-04
3	3.03E-04
4	3.09E-04
5	3.08E-04
6	3.38E-04
7	3.07E-04
8	3.19E-04
9	9.14E-02
10	6.53E-04

Tabla 3. Tiempos para obtener la ruta más corta, maze 7 * 7

Numero de Experimento	Tiempo
1	7.88E+01
2	1.76E+02
3	6.09E+01
4	1.48E+02
5	1.28E+02
6	7.65E+01
7	1.20E+02
8	9.36E+01
9	5.20E+01
10	9.96E+01

Tabla 4. Tiempos para recorrer la trayectoria MCL, maze 7 * 7

Otro experimento se realizó con un Maze de 10 * 10 en donde el nodo inicial es 6 y el nodo final es el 95. Figura 48.

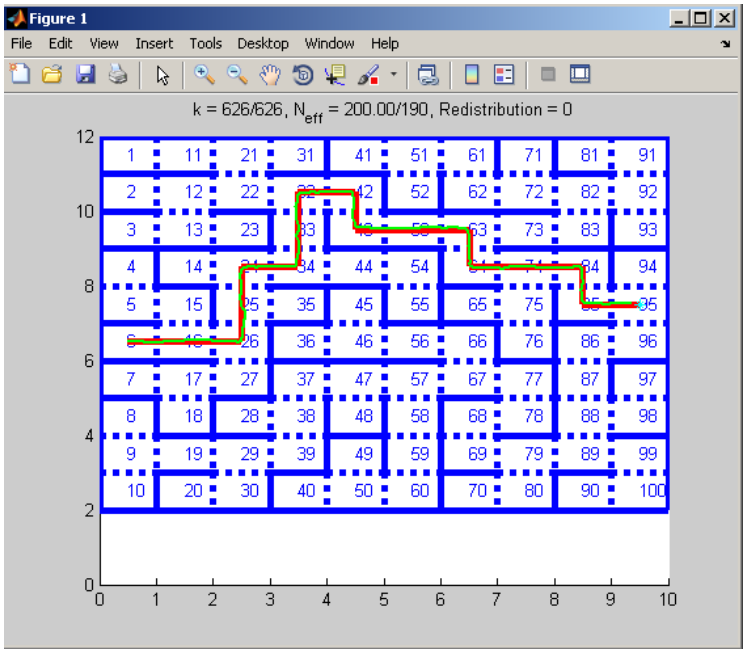

Figura 48. Maze y recorrido del robot. Maze 10 * 10.

La Figura 49. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **6** al nodo final **95**.

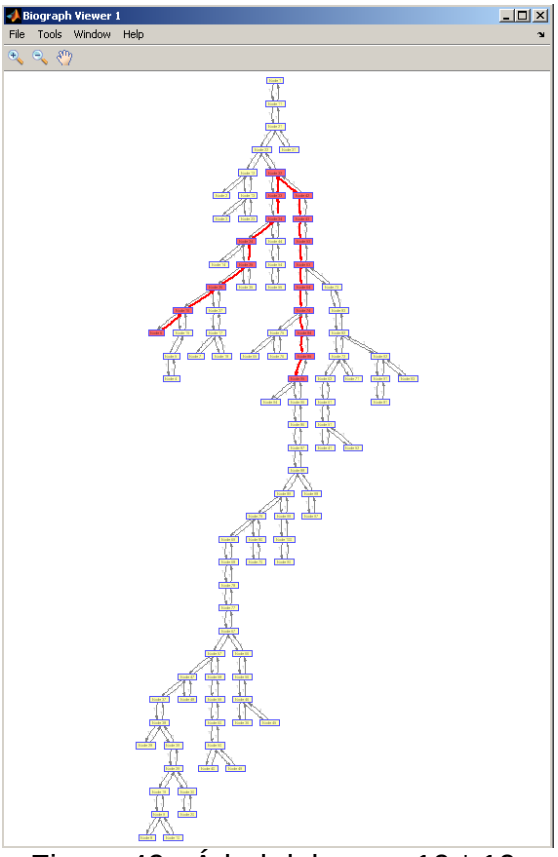

Figura 49. Árbol del maze 10 * 10.

El tiempo obtenido en la ruta más corta fue 0.00040368259 y el tipo obtenido en MCL fue 174.13922.

A continuación se muestran los tiempos obtenidos con un Maze de 10 * 10.

Numero de Experimento	Tiempo
1	4.33E-04
2	5.57E-04
3	5.01E-04
4	4.43E-04
5	5.32E-04
6	5.10E-04
7	4.70E-04
8	5.17E-04
9	5.30E-04
10	5.43E-04

Tabla 5. Tiempos para obtener la ruta más corta, maze 10 * 10

Numero de Experimento	Tiempo
1	2.86E+02
2	3.50E+02
3	1.47E+02
4	1.04E+02
5	4.80E+02
6	2.00E+02
7	1.57E+02
8	3.96E+02
9	4.73E+02
10	1.28E+03

Tabla 6. Tiempos para recorrer la trayectoria MCL, maze 10 * 10

Otro experimento se realizó con un Maze de 15 * 15 en donde el nodo inicial es **4** y el nodo final es el **220**. Figura 50.

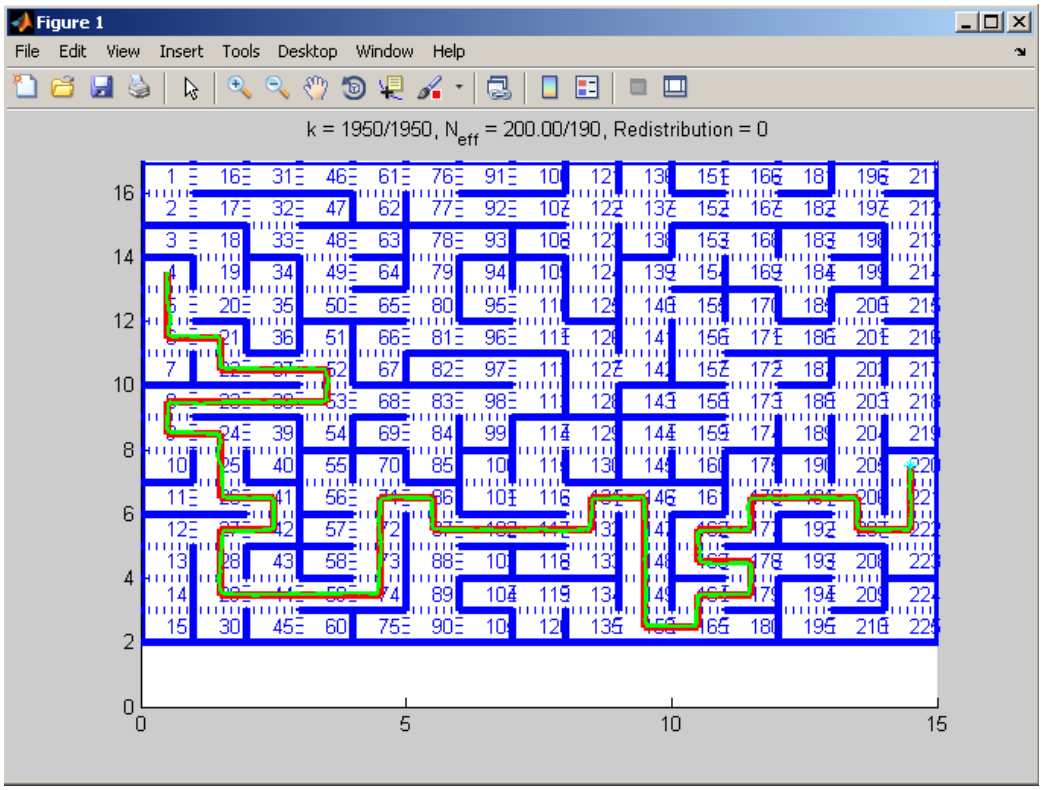

Figura 50. Maze y recorrido del robot. Maze 15 * 15.

La Figura 51. Muestra el Árbol del Maze y se observa el recorrido desde el nodo inicial **4** al nodo final **220**.

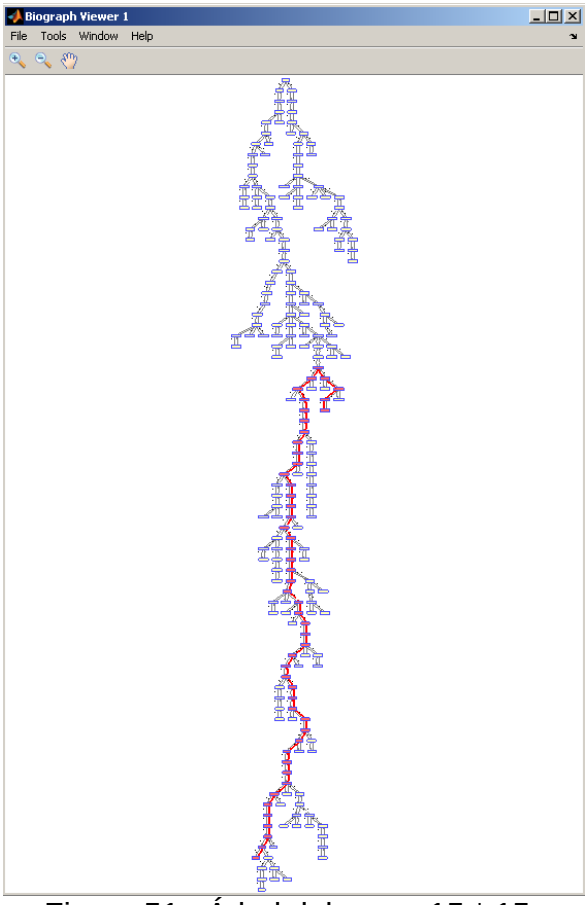

Figura 51. Árbol del maze 15 * 15.

El tiempo obtenido en la ruta más corta fue 0.11643295 y el tipo obtenido en MCL fue 3267.9092.

A continuación se muestran los tiempos obtenidos con un Maze de 15 * 15.

Numero de Experimento	Tiempo
1	1.33E-01
2	9.01E-04
3	6.16E-04
4	6.44E-04
5	1.86E-02
6	9.47E-04
7	6.38E-04
8	6.00E-04
9	6.34E-04
10	1.45E-01

Tabla 7. Tiempos para obtener la ruta más corta, maze 15 * 15

Numero de Experimento	Tiempo
1	3.04E+03
2	5.40E+02
3	4.57E+02
4	7.63E+02
5	3.06E+03
6	5.89E+02
7	2.23E+03
8	1.77E+03
9	9.31E+02
10	3.06E+03

Tabla 8. Tiempos para recorrer la trayectoria MCL, maze 15 * 15

Capítulo 4

Conclusiones

4. CONCLUSIONES

4.1 Conclusiones y Trabajo Futuro.

En este trabajo se presenta una metodología para la solución al problema de la navegación de un robot móvil. Primero, se generaron varios Mazes perfectos de forma al azar. Estos Mazes fueron generados para mostrar el ambiente por el cual el Robot Móvil tiene que resolver el problema de navegación. En la medida que el Maze es de mayor tamaño se vuelve un reto encontrar la salida. Por esa razón, es importante contar con un algoritmo que encuentre la salida y obtenga la ruta más corta. Un Maze puede ser representado como un grafo y utilizar la teoría de grafos para su solución. De acuerdo a las características del Maze se implementó el algoritmo de Dijkstra para obtener la ruta más corta. Este algoritmo de complejidad O(log(N)*E) se implementó obteniendo resultados muy favorables y que permitieron en esta investigación obtener una planeación de la ruta por donde el Robot móvil navegara. Una vez obtenida la ruta más corta se implementó el algoritmo de Monte Carlo localización también conocido como filtrado de partículas logrando que el Robot Navegara por una ruta previamente definida.

Como parte de trabajos futuros se pretende Implementar estos algoritmos en un Robot móvil y comparar otros métodos para la obtención de la ruta más corta. Continuar investigando sobre las mejoras realizadas al Método de Montecarlo Localización e implementarlas para lograr que los tiempos sean reducidos. Otra línea de desarrollo es considerar en el laberinto los sentidos (como si fuera una carretera en donde existen los sentidos vehiculares) y encontrar la ruta más corta considerando esos sentidos. También como parte de trabajos futuros se considera diseñar mazes dinámicos que cambien conforme se vaya recorriendo el maze y utilizar robots multiagentes que compartan información para encontrar la ruta más corta.

Referencias Bibliográficas:

[1] Almuallim,Hussein;Kaneda,Shigeo;Akiba,Yasuhiro,*Developmentand applications of decision trees*,Expert Systems, Vol. 1,Academic Press,2002.

[2] Autere,Antti, *Extensionsand Applications of the A* Algorithm*,Helsinki University of Technology Laboratory for Theoretical Comp Sci.,2005.

[3] Becker,Marcelo; Hall,Richard; Jensen,Björn; Kolski,Sascha; Maček,Kristijan; Siegwart,Roland, *The use of Obstacle Motion Tracking for Car-like Mobile Robots Collision Avoidance in Dynamic Urban Environments*,Proceedingsof the XII International Symposium on Dynamic Problems of Mechanics,SP, Brazil,2007.

[4] Berglund,Tomas;Brodnik,Andrej;Jonsson,Håkan; Fanson,Mats; Söderkvist,Inge, *planning smooth and obstacle - Avoiding B-spline paths for autonomous mining vehicles*,IEEE Transactions on Automation Science and Engineering,Vol. 7, No. 1,2010.

[5] Bruggemann,Bernd; Kamphans,Tom; Langetepe,Elmar, *Escaping from a Labyrinth with One-way Roads for Limited Robots*,Comp. Sci.,University of Bonn, Bonn, Germany,2009.

[6] Cocora,Alexandru;Kersting,Kristian; Plagemanny,Christian; Burgardy, Wolfram; De Raedt,Luc, *Learning Relational Navigation Policies*,University of Freiburg. Freiburg,Germany,2006.

[7] Cowan,G., *Monte Carlo Techniques*,2009.

[8] Dellaert,Frank,*ASample of Monte Carlo Methods in Robotics and Vision*,Collegeof Computing, Georgia Institute of Technology,Atlanta, GA,2008.

[9] Dellaert,Frank; Seitz,Steven M.; Thorpe,Charles E.; Thrun,Sebastian, *EM,MCMC,and Chain Flipping for Structure fromMotion with Unknown Correspondence*,Comp Sci., Department, Carnegie Mellon University,Pittsburgh PA,2000.

[10] Elinas,Pantelis,*On the Design and Implementation of Decision-Theoretic, Interactive, and Vision-Driven Mobile Robots*,Comp. Sci.,The University of British Columbia,2008.

[11] Huang,Timothy; Nevmyvaka,Yuriy, *A Practical Markov Chain Monte Carlo Approach to Decision Problems*,Comp. Sci.,Middlebury College,2008.

[12] Kapoor,S.; Ramesh,H., *An Algorithm for Enumerating All Spanning Trees of a Directed Graph1*,Springer-Verlag, New York Inc.,2000.

[13] Oh,Songhwai; Russell,Stuart; Sastry,Shankar, *Markov chain Monte Carlo Data Association for General Multiple Traget Tracking Problems*,2005.

[14] Rohrmuller,Florian; Althoff,Matthias; Wollherr,Dirk;Buss,Martin, *Probabilistic Mapping of Dynamic Obstacles Using Markov Chains for Replanningin Dynamic Environments*,IEEE/RSJ International Conference on Intelligent Robots and Systems,Nice, France, Sept, 22-26, 2008

[15] Schenker,P. S.; Huntsberger,T. L.; Pirjanian,P.; Baumgartner, E.;Aghazarian,H.;Trebi-Ollennu,A.;Leger,P. C.; Cheng,Y.; Backes,P. G.;Tunstel,E. W.; Dubowsky,S.,*Robotic automation for space: planetary surfaceexploration, terrain-adaptive mobility, and multi-robot cooperative tasks*,California Instituteof Technology,Pasadena, California,2007.

[16] Stamenova, Stiliyana, *Solving the Maze: Robot Localization Using the Monte Carlo Localization Algorithm and Shape Context*, Macalester College, 2009.

[17] Thrun,Sebastian;Fox,Dieter;Burgard,Wolfram;Dellaert,Frank,*Robust Monte Carlo Localization for Mobile Robots*,Comp. Sci.,Carnegie Mellon University,Pittsburgh, PA,2001.

[18] Tingle,D; Ball,E; Augat,M.,*Maze Solving by Learning State topologies*,Comp. Sci.,Swarthmore College,2009.

[19] Turner,MiltonJ., *Obstacle avoidance and path traversal using interactive machine learning*,Brigham Young University,2007.

[20] Ye Wu,Bang; Chao,Kun-Mao, *Spanning Trees and Optimization Problems (Excerpt)*,2008.

[21] Thrun, Sebastian; Fox, Dieter; Burgard, Wolfram; Dellaert, Frank, *Efficient Position Estimation for Mobile Robots*, Comp. Sci., Carnegie Mellon University, Pittsburgh, PA, 2000.

[22] Grimaldi, Ralph p; *Matemáticas Discreta y Combinatoria*, Quinta Edición, Addison Wesley, 2003.

[23] Díaz, Carmen; *Dificultades en la resolución de problemas que involucran el Teorema de Bayes*,Universidad de Granada, España. 2007.

[24] Chaudhuri ,SheliSinha; Konar ,Amit; *Vision Based Target-Tracking Realized with Mobile Robots using Extendes Kalmar Filter*,Department of Jadavpur University, India. 2007.

[25] Mäkelä ,Hannu; *OUTDOOR NAVIGATION OF MOBILE ROBOTS*,Helsinki University of Technology, Finland. 2001.

[26] Fouque,Clément;Bonnifait, Philippe;Bétaille,David; *Enhancement of GlobalVehicle LocalizationusingNavigableRoad Maps and Dead-Reckoning*, Université de Technologie de Compiègne, France. 2008.

[27] Chung,Hakyoung;Ojeda,Lauro;Borenstein,Johann; *AccurateMobileRobotDead-reckoning With a Precision-calibrates Fiber Optic Gyroscope*, IEEE Transactions on Robotics and Automation, Vol. 17, No.1, February, 80-84. 2001.

[28] Guivant,Jose;Nebot,Eduardo;Baiker,Stephan; *Autonomous Navigation and Map building Using Laser Range Sensors in Outdoor Applications*, Journal of Robotic Systems, Vol. 17, No. 10, October, 565-583. 2000.

APENDICE A. Introducción a Grafos

Definición 1. Un *grafo* G = (*V,E*) es una estructura combinatoria constituida por un conjunto V=V(G) de elementos llamados *vértices* y un conjunto E=E(G) de pares no ordenados de vértices distintos llamados *aristas*[22].

Si la arista *e* = {*u,v*} = *uv* relaciona los vértices *u* y *v* , se dice que *u* y *v*son vértices adyacentes. Las aristas *e=uv*y *f* = *wz* son *aristas independientes* si no tienen vértices en común, es decir {u,v} ∪ {w,z} = ∅ El número de vértices de G, |V(G)|, es el *orden* del grafo y el número de aristas |E(G)| es su *tamaño*. Por ejemplo, en el grafo representado en la Figura 1, de orden 5 y tamaño 7, los vértices *u* y *v* son adyacentes mientras que *u* y *w* son vértices independientes [22].

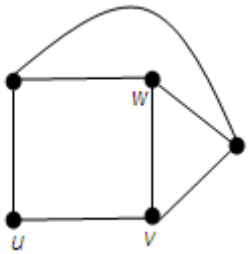

Figura 52. Grafo con orden 5 y tamaño 7.

Las aristas que unen un vértice con él mismo, se denomina *ciclo* y las *aristas paralelas* son aquella que unen un mismo par de vértices. Figura 2.

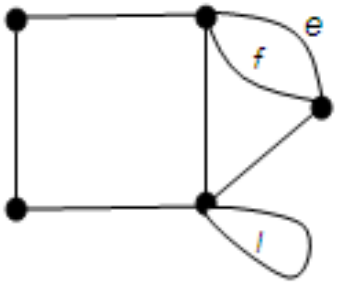

Figura 53. Grafo con un ciclo *l* y aristas paralelas *e* y f.

Definición 2. Sean x, y vértices (distintos) de un grafo no dirigido G = (V,E). Un camino x-y en G es una sucesión alternada finita (sin ciclos)

$$x = x_o, e_1, x_1, e_2, x_2, e_3, \ldots, e_{n-1}, x_{n-1}, e_n, x_n = y$$

de vértices y aristas de G, que comienza en el vértice *x* y termina en el vértice *y* y que contiene las *n* aristas $e_i = \{x_{i-1}, x_i\}$ donde $1 \leq i \leq n$.

La *longitud* de este camino es *n*, el número de aristas que contiene.

Cualquier camino x-y donde x = y (y n>1) es un *camino cerrado*, en caso contrario el camino es *abierto*.

Cuando ningún vértice del camino x-y se presenta más de una vez, el camino es un *camino simple* x-y.

La dirección de una arista se indica al colocar una flecha dirigida sobre ella.

Definición 3. Sea G = (V,E) un grafo no dirigido. Decimos que G es *conexo* si existe un camino simple entre cualesquiera dos vértices distintos de G.

Sea G = (V,E) un grafo dirigido. Su grafo no dirigido asociado es el grafo obtenido de G si no se tiene en cuenta las direcciones de las aristas. Si se obtiene más de una arista no dirigida de un par de vértices distintos de G, entonces sólo una de estas aristas se dibuja en el grafo no dirigido asociado. Cuando este grafo asociado es conexo, consideramos que G es conexo.

Un grafo que no es conexo es *disconexo*.

Definición 4. Sea G = (V,E) un grafo no dirigido sin ciclos. El grafo G es un *árbol* si G es conexo y no contiene ciclos.

Si un grafo es un árbol, escribimos T en vez de G, para enfatizar esta estructura.

Teorema 1. Si a, b son vértices distintos de un árbol T, entonces hay un camino único que conecta estos vértices.

El *grado* de un vértice *u*es el número de vértices adyacentes a *u*.

Definición 5. *Árbol binario* es aquel en donde existe un único vértice r, llamado raíz, que tiene grado 2 y los vértices restantes tienen grado 1 o 3. Figura 3.

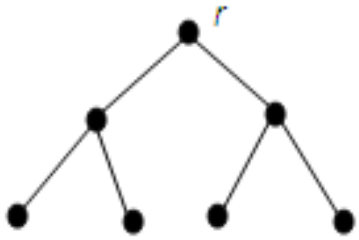

Figura 54. Árbol binario.

APENDICE B. Teorema de Bayes

El cálculo de probabilidades condicionales inversas mediante el teorema de Bayes es fundamental en las aplicaciones de la Estadística, porque permite incorporar cambios en nuestro grado de creencia sobre los sucesos aleatorios, a medida que adquirimos nueva información. Este tipo de razonamiento es muy importante en tareas profesionales, como el diagnóstico, evaluación, toma de decisiones y aplicación de la inferencia estadística en la investigación empírica.[23]

El Teorema de Bayes, dentro de la teoría probabilística, proporciona la distribución de probabilidad condicional de un evento "A" dado otro evento "B" (probabilidad posteriori), en función de la distribución de probabilidad condicional del evento "B" dado "A" y de la distribución de probabilidad marginal del evento "A" (probabilidad simple o apriori).

Partiendo de las fórmulas de probabilidad condicional $P(A/B) = \frac{P(A \cap B)}{P(B)}$ y probabilidad conjunta $P(A \cap B) = P(B)\, P(A/B)$ para eventos estadísticamente dependientes se procederá a enunciar el Teorema de Bayes.

Sean $\boldsymbol{A_1}, \boldsymbol{A_2} \ldots \boldsymbol{A_k}$ eventos mutuamente excluyentes tales que, cualquier evento "B" en el espacio muestral pertenece a uno y sólo a uno de estos eventos. Entonces la probabilidad de que ocurra cualquier evento $\boldsymbol{A_k}$ dado que ha ocurrido el evento "B" se calculará por la siguiente fórmula:

$$P\left(A_k/B\right) = \frac{P(A_k \cap B)}{P(B)}$$

Por lo tanto, sustituyendo la fórmula de probabilidad condicional, se obtiene la fórmula general para el Teorema de Bayes:

$$P\left(A_k/_B\right) = \frac{P(A_k)\,P(B/A_k)}{P(A_1)\,P(B/A_1) + P(A_2)\,P(B/A_2) + \ldots + P(A_k)\,P(B/A_k)}$$

Dónde:

- El numerador es la probabilidad conjunta:

$$P(A \cap B) = P(B)\,P\left(A/_B\right)$$

- El denominador es la probabilidad marginal de que ocurra el evento “B”

$$P(B) = P(A_1)\,P(B/A_1) + P(A_2)\,P(B/A_2) + \ldots + P(A_k)\,P(B/A_k)$$

Ejemplo:

Una fábrica produce un artículo en tres diferentes máquinas. Del total de la producción el 30% es producido en la maquina A, en 50% en la maquina B y el 20% lo produce la maquina C. La probabilidad de que un artículo producido por una maquina específica sea de primera calidad, se muestra en la siguiente tabla:

Maquina	Probabilidad
A	0.8
B	0.7
C	0.9

Si se selecciona un artículo aleatoriamente de la línea de producción:

a) ¿Cuál es la probabilidad de que sea de primera calidad?
b) Si el articulo seleccionado es de primera calidad, ¿cuál es la probabilidad de que haya sido producido por la maquina A?

Solución:

$P(A)$= 0.3 $P\left(Q/_A\right)$= 0.8

$P(B)$= 0.5 $P\left(Q/_B\right)$= 0.7

$P(C)$= 0.2 $P\left(Q/_C\right)$= 0.9

$$P(A \cap Q) = P(A)\ P\left(Q/_A\right) = (0.3)(0.8) = 0.24$$
$$P(B \cap Q) = P(B)\ P\left(Q/_B\right) = (0.5)(0.7) = 0.35$$
$$P(C \cap Q) = P(C)\ P\left(Q/_C\right) = (0.2)(0.9) = 0.18$$

a) La probabilidad de que sea de primera calidad es la siguiente:

$$P(Q) = P(A)\ P(Q/A) + P(B)\ P(Q/B) + P(C)\ P(Q/C) = 0.24 + 0.35 + 0.18 = \mathbf{0.77}$$

b) La probabilidad de que haya sido producido por la maquina A, es la siguiente:

$$P(A \cap Q) = \frac{0.24}{0.77} = \mathbf{0.31}$$

Desarrollo de un algoritmo híbrido Dijkstra - Montecarlo para la planeación y optimización de rutas en un robot móvil

Aceves-Fernandez Marco Antonio, Salinas-González Eric Francisco, Gorrostieta-Hurtado Efren, Pedraza-Ortega J. Carlos, Ramos Arreguín Juan Manuel, Tovar Arriaga Saúl

Universidad Autónoma de Querétaro, Fac. de Informática
Campus Juriquilla, Av. de las Ciencias S/N, Juriquilla,
Delegación Santa Rosa Jauregui, CP 76230, Querétaro, México
marco.aceves@uaq.mx

Resumen

Se presenta un algoritmo híbrido para la planeación y optimización de rutas en un robot móvil. Este algoritmo se desarrolló utilizando una técnica basada en el algoritmo de Dijkstra para la planeación de rutas y el algoritmo de localización de Montecarlo (Montecarlo Localization ó MCL) para optimizar la ruta. Se realizaron pruebas bajo diversos escenarios de laberintos y se mostró que las pruebas fueron consistentes en todos los escenarios. Estas pruebas, demostraron la robustez del método planteado como una alternativa a los métodos existentes de planeación de rutas en robots móviles.

Palabras clave: Monte Carlo, Algoritmo Dijkstra, Robot Móvil, planeación de rutas.

1. Introducción

El principal objetivo de la robótica es la construcción de máquinas capaces de realizar tareas con la flexibilidad, la robustez y la eficiencia que exhiben los seres humanos. Los robots son potencialmente útiles en escenarios peligrosos para el ser humano, sucios o difíciles[1].

El robot móvil se caracteriza por realizar una serie de desplazamientos (navegación) y por llevar a cabo una interacción con distintos elementos de su entorno de trabajo (operación), que implican el cumplimiento de una serie de objetivos impuestos según cierta especificación.

Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno a través de sus sensores, de modo que le permita crear una abstracción del mundo; la planificación de una trayectoria libre de obstáculos y el guiado del vehículo a través de la referencia construida. De forma simultánea, el vehículo puede interaccionar con ciertos elementos del entorno. Así, se define el concepto de operación como la programación de las herramientas de a bordo que le permiten realizar la tarea especificada.

Muchos algoritmos que construyen mapas en tiempo de ejecución, necesitan robots con suficiente memoria, poder de procesamiento o técnicas de sensores avanzados[2]. Sin embargo, en muchas ocasiones es necesario utilizar robots limitados por razones de costo y espacio.

Existen diversas técnicas como lo son: Dead-Reckoning, Cadenas de Markov, redes de Bayes o Filtro de Kalman, Node Combination, entre otras [3-8]

El algoritmo de localización de Markov es más exacto que el filtro de Kalman y puede ser utilizado para una localización global. Sin embargo la localización de Markov es difícil de implementar. [5]

2. Algoritmo Dijkstra

El algoritmo Dijkstra se utilizará en esta contribución para obtener la ruta más corta del lugar origen al lugar destino [9].

Se comienza con un grafo dirigido conexo sin ciclos G. A cada arista e = (a, b) de este se le asigna un número real positivo llamado el peso de e, que se denota con w(e) o w(a, b). Si $x,y \in V$ pero $(x,y) \notin E$, se define que $w(x,y) = \infty$.

Cuando se da un grafo G con las asignaciones de peso aquí descritas, se dice que el grafo es un grafo ponderado.

Dado G, para cada $e = (a,b) \in E$ se interpreta p(e) como la longitud de una ruta directa de a a b. Para cualesquiera dos vértices $a,b \in V$ se escribe d(a,b) como la distancia (más corta) de a a b. Si no existe

tal camino (en G) de aab entonces se define d(a,b) = ∞, Para cualquier a $\in$ V, d(a,a) = 0.

Se Fija ahora v0 $\in$ V. Entonces para cualquier v $\in$ V, se determina d(v0,v) un camino simple dirigido de v0 a v si d(v0,v) es finito.

Sea | V | = n. Para determinar la distancia más corta de un vértice fijo v0 a los demás vértices de G, así como un camino simple dirigido más corto para cada uno de estos vértices, se aplica el siguiente algoritmo.

Paso 1: Se inicializa i = 0 y S0 = {v0}. Se etiqueta v0 con (0,—) y cada v ≠ v0 con (∞ , —)

Paso 2: Para cada v $\in$ Si, se reemplaza, la etiqueta de v por la nueva etiqueta final (L(v),y), donde

$$L(v) = \text{mín}\ \{L(v), L(u) + p(u,v)\},\ U \in Si \qquad (1)$$

Y y es un vértice en Si que produce el L(v) mínimo. Si se hace un reemplazo, esto se debe al hecho de que se puede ir de v0 a v y recorrer una distancia más corta si se recorre un camino que incluye una arista (y,v).

Paso 3: Si cada vértice de Si (para algún $0 \leq i \leq n-2$) tiene la etiqueta (∞ , —) , entonces el grafo etiquetado contiene la información del camino más corto.

Si no, existe al menos un vértice v $\in$ Si que no está etiquetado como (∞ , —) y se realizan las siguientes tareas:

Se selecciona un vértice vi+1, tal que L(vi+1) sea mínimo. Puede haber varios de estos vértices cuyo caso se elige alguno de los posibles candidatos. El vértice vi+1 es un elemento de Si que es el más cercano a v0.

Se actualiza i= i+1.

Si i = n – 1, el grafo etiquetado contiene la información del camino más corto. Se regresa al paso 2.

3. Localización Montecarlo (MCL)

Localización Monte Carlo (MCL) es una técnica estadística para la localización del robot basada en la localización de Markov. Como en la localización de Markov, se requiere un mapa del entorno del Robot. Esto es un filtro de Bayes [10].

Localización Monte Carlo (MCL) fue desarrollado por Dellaert y Fox [11] Ellos tomaron el método de filtrado de partículas utilizado en otras áreas como visión por computadora. Localización Monte Carlo es uno de los métodos más populares en la localización, porque ofrece soluciones a una gran variedad de problemas.

La idea principal del filtro de Bayes es estimar una densidad de probabilidad sobre el estado espaciado condicionado sobre los datos. Este posterior que se suele llamar la creencia (Bel) y se denota

$$Bel(xt) = p(xt|\ d0...t) \qquad (2)$$

En la ecuación 2, x denota el estado, xt es el estado en el tiempo t y d0, t denota el tiempo inicial desde 0 a t. Denotando o (observación) y a (acción).

El modelo de movimiento, p(x‘|x,a), es una generalización probabilística de la cinemática del robot. Para un robot que operan en el plano las posiciones de x y x’ son variables en tres dimensiones.

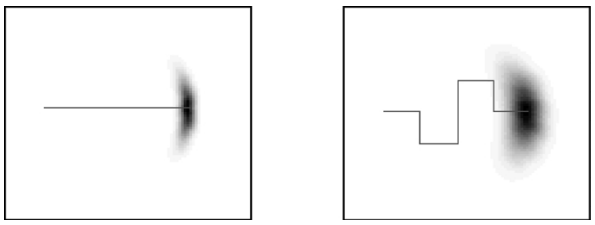

Fig. 1. Densidad p(x‘|x,a) después de mover 40 metros (diagrama izquierdo) y 80 metros (diagrama de la derecha).

En el ejemplo de la figura 1, la postura inicial de x se muestra a la izquierda, y la línea continua representa los datos de odometría, medido por el robot. El área sombreada de la derecha muestra la densidad p(x‘|x,a): la posición más oscura, es la que tiene mayor probabilidad. Una comparación de los dos diagramas revela que el margen de incertidumbre depende del movimiento global: A pesar de que la postura de un robot libre de ruido es la misma para ambos segmentos de movilidad, la incertidumbre en el diagrama de la derecha es más grande debido a la mayor distancia recorrida por el robot[11].

Un modelo de toma de muestras es una rutina que acepta x y a como entrada y genera al azar posiciones x’ distribuidas de acuerdo a p(x‘|x,a). La secuencia de conjuntos de partículas se aproxima a la densidad de un robot que mide sólo odometría.

4. Resultados Experimentales

Para poder modelar el comportamiento de este algoritmo: Dijkstra - Montecarlo, se construye el ambiente por el cual Robot tendrá que navegar. En éste laberinto de n * n (figura 2).el robot navegará desde el inicio(nodo 1) hasta alcanzar la meta (en este caso el nodo 25).

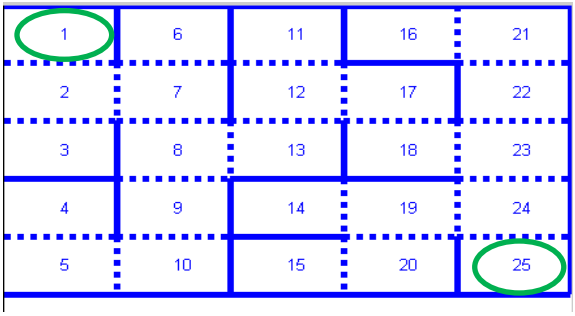

Fig. 2. Generación del Laberinto

En la Figura 2 se muestran las paredes (líneas solidas), mientras que las puertas por donde el Robot puede navegar hasta encontrar su objetivo se muestra mediante las líneas punteadas. Para encontrar el camino más corto dentro del laberinto, se utilizara el algoritmo de Dijkstra y la teoría de grafos. En la figura 3 se muestra el camino que siguió el robot navegando en el laberinto de la figura 2.

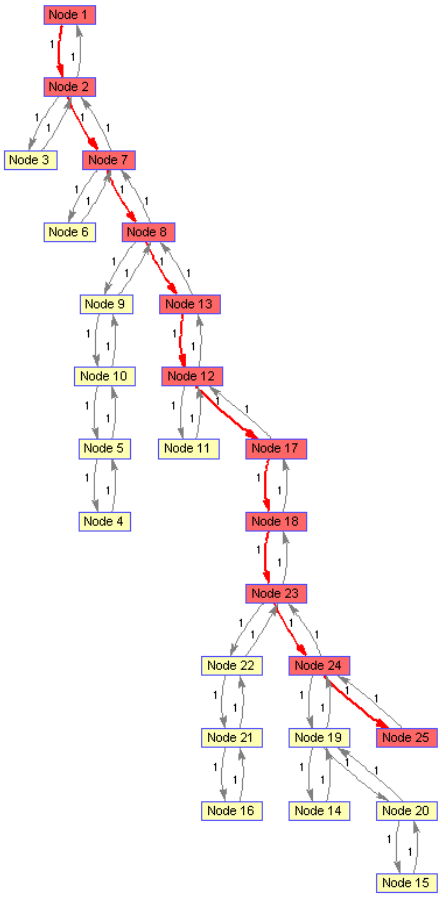

Fig. 3. Cálculo de la ruta mas corta mediante el algoritmo Dijkstra.

En el caso de la figura 3, la ruta más corta es; 1,2,7,8,13,12,17,18,23,24,25 . Una vez obtenida se traza la trayectoria por la cual navegara el Robot. La simulación se realiza por medio del algoritmo de Montecarlo. Esto se muestra en la figura 4.

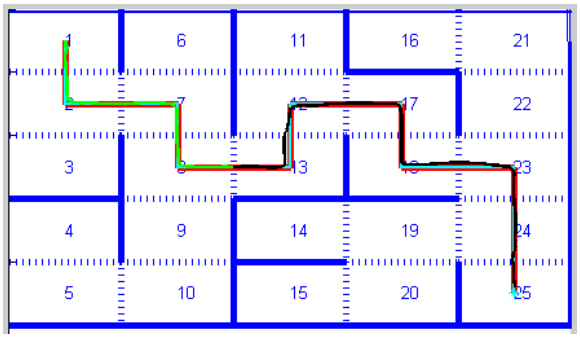

Fig. 4. Simulación de la navegación del Robot en una ruta establecida mediante Localización Monte Carlo.

Para un laberinto de 5 x 5 como el que se muestra en la figura 4, se realizaron diversos experimentos para demostrar que sin importar la posición de inicio o final, el algoritmo Dijkstra – MCL obtiene la ruta mas corto en un tiempo relativamente corto. Los tiempos para obtener la ruta mas corta y para recorrer la trayectoria se muestran en las tablas 1 y 2, respectivamente.

No. de Exp.	Tiempo
1	8.14E-04
2	9.62E-04
3	3.96E-04
4	4.68E-04
5	7.73E-04
6	1.16E-01
7	2.51E-04
8	2.47E-04
9	2.50E-04
10	2.52E-04

Tabla 1. Resultados experimentales de un laberinto de 5x5 para obtener la ruta mas corta.

Como puede verse en la tabla 1, los tiempos para poder calcular la ruta mas corta son consistentes, aunque depende el tiempo del número de nodos a recorrer. Además, se realizaron experimentos con laberintos de 6x6, 7x7 y 10x10, utilizando la metodología citada. La tabla 2 muestra los resultados de un laberinto 10x10.

No. de Exp.	Tiempo
1	2.94E+01
2	3.07E+01
3	9.04E+01
4	4.20E+01

5	1.22E+01
6	8.84E+01
7	1.52E+02
8	2.47E-04
9	3.56E+01
10	4.88E+01

Tabla 2. Resultados experimentales de un laberinto de 10x10 para obtener la ruta mas corta.

5. Conclusiones y trabajo futuro

Se ha mostrado que el algoritmo Dijkstra – MCL es útil para la planeación y optimización de rutas para robots móviles. Este algoritmo ha demostrado consistencia en diferentes escenarios de laberintos, por lo que puede ser usado en aplicaciones donde se necesita robustez y consistencia.
Como trabajo futuro se sugiere la implementación de este algoritmo en software embebido para implementarse en un robot móvil.

Referencias

[1] Tingle, D; Ball E; Augat, M., **Maze Solving by Learning State topologies**, Comp. Sci., Swarthmore College, 2009.

[2] Bruggemann, Bernd; Kamphans, Tom; Langetepe, Elmar, **Escaping from a Labyrinth with One-way Roads for Limited Robots**, Comp. Sci., University of Bonn, Bonn, Germany,2009.

[3] Rohrmuller, Florian; Althoff, Matthias; Wollherr, Dirk; Buss, Martin, **Probabilistic Mapping of Dynamic Obstacles Using Markov Chains for Replanning in Dynamic Environments**, IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, Sept, 22-26, 2008.

[4] Widodo Budiharto, Djoko Purwanto and Achmad Jazidie, **A robust obstacle avoidance for service robot using Bayesian approach**, International Journal of Advanced Robotic Systems, Vol. 8, No. 1, ISSN 1729-8806, pp 37-44, 2011.

[5] Ghahraman Zoubin, **An introduction to Hidden Markov Models and Bayesian**, International Journal of Pattern Recognition and Artificial Intelligence, no. 15, vol. 1, pp. 942, 2001.

[6] Rong-Jong Wai • Chia-Ming Liu • You-Wei Lin, **Robust path tracking control of mobile robot via dynamic petri recurrent fuzzy neural network**, Soft Comput., no 15, pp. 743–767, 2011.

[7] Yanrui Geng Æ Jinling Wang, **Adaptive estimation of multiple fading factors in Kalman filter for navigation applications**, GPS Solut., vol. 12, pp. 273–279, 2008.

[8] Xin Lu, Martin Camitz, **Finding the shortest paths by node combination**, Applied Mathematics and Computation, vol. 217, pp.6401–6408, 2011.

[9] Youming Li, Ardian Greca, and James Harris, **On Dijkstra's Algorithm for Deadlock Detection**, Advanced Techniques in Computing Sciences and Software Engineering, pp. 385-387, 2010.

[10] Stamenova, Stiliyana, **Solving the Maze: Robot Localization Using the Monte Carlo Localization Algorithm and Shape Context**, Macalester College, Ph.D. Thesis 2009.

[11] Thrun,Sebastian;Fox,Dieter;Burgard,Wolfram;Dellaert,Frank,**Robust Monte Carlo Localization for Mobile Robots**, Comp. Sci., Carnegie Mellon University ,Pittsburgh, PA,2001.

Desarrollo de un algoritmo híbrido Dijkstra - Montecarlo para la planeación y optimización de rutas en un robot móvil

Aceves-Fernandez Marco Antonio, Salinas-González Eric Francisco, Gorrostieta-Hurtado Efren, Pedraza-Ortega J. Carlos, Ramos Arreguín Juan Manuel, Tovar Arriaga Saúl

Universidad Autónoma de Querétaro, Fac. de Informática
Campus Juriquilla, Av. de las Ciencias S/N, Juriquilla,
Delegación Santa Rosa Jauregui, CP 76230, Querétaro, México
marco.aceves@uaq.mx

Resumen

Se presenta un algoritmo híbrido para la planeación y optimización de rutas en un robot móvil. Este algoritmo se desarrolló utilizando una técnica basada en el algoritmo de Dijkstra para la planeación de rutas y el algoritmo de localización de Montecarlo (Montecarlo Localization ó MCL) para optimizar la ruta. Se realizaron pruebas bajo diversos escenarios de laberintos y se mostró que las pruebas fueron consistentes en todos los escenarios. Estas pruebas, demostraron la robustez del método planteado como una alternativa a los métodos existentes de planeación de rutas en robots móviles.

Palabras clave: Monte Carlo, Algoritmo Dijkstra, Robot Móvil, planeación de rutas.

1. Introducción

El principal objetivo de la robótica es la construcción de máquinas capaces de realizar tareas con la flexibilidad, la robustez y la eficiencia que exhiben los seres humanos. Los robots son potencialmente útiles en escenarios peligrosos para el ser humano, sucios o difíciles[1].

El robot móvil se caracteriza por realizar una serie de desplazamientos (navegación) y por llevar a cabo una interacción con distintos elementos de su entorno de trabajo (operación), que implican el cumplimiento de una serie de objetivos impuestos según cierta especificación.

Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno a través de sus sensores, de modo que le permita crear una abstracción del mundo; la planificación de una trayectoria libre de obstáculos y el guiado del vehículo a través de la referencia construida. De forma simultánea, el vehículo puede interaccionar con ciertos elementos del entorno. Así, se define el concepto de operación como la programación de las herramientas de a bordo que le permiten realizar la tarea especificada.

Muchos algoritmos que construyen mapas en tiempo de ejecución, necesitan robots con suficiente memoria, poder de procesamiento o técnicas de sensores avanzados[2]. Sin embargo, en muchas ocasiones es necesario utilizar robots limitados por razones de costo y espacio.

Existen diversas técnicas como lo son: Dead-Reckoning, Cadenas de Markov, redes de Bayes o Filtro de Kalman, Node Combination, entre otras [3-8]

El algoritmo de localización de Markov es más exacto que el filtro de Kalman y puede ser utilizado para una localización global. Sin embargo la localización de Markov es difícil de implementar. [5]

2. Algoritmo Dijkstra

El algoritmo Dijkstra se utilizará en esta contribución para obtener la ruta más corta del lugar origen al lugar destino [9].

Se comienza con un grafo dirigido conexo sin ciclos G. A cada arista e = (a, b) de este se le asigna un número real positivo llamado el peso de e, que se denota con w(e) o w(a, b). Si $x,y \in V$ pero $(x,y) \notin E$, se define que $w(x,y) = \infty$.

Cuando se da un grafo G con las asignaciones de peso aquí descritas, se dice que el grafo es un grafo ponderado.

Dado G, para cada $e = (a,b) \in E$ se interpreta p(e) como la longitud de una ruta directa de a a b. Para cualesquiera dos vértices $a,b \in V$ se escribe d(a,b) como la distancia (más corta) de a a b. Si no existe

tal camino (en G) de aab entonces se define d(a,b) = ∞, Para cualquier a $\in$ V, d(a,a) = 0.

Se Fija ahora v0 $\in$ V. Entonces para cualquier v $\in$ V, se determina d(v0,v) un camino simple dirigido de v0 a v si d(v0,v) es finito.

Sea | V | = n. Para determinar la distancia más corta de un vértice fijo v0 a los demás vértices de G, así como un camino simple dirigido más corto para cada uno de estos vértices, se aplica el siguiente algoritmo.

Paso 1: Se inicializa i = 0 y S0 = {v0}. Se etiqueta v0 con (0,—) y cada v ≠ v0 con (∞ , —)

Paso 2: Para cada v $\in$ Si, se reemplaza, la etiqueta de v por la nueva etiqueta final (L(v),y), donde

$$L(v) = \text{mín}\ \{L(v), L(u) + p(u,v)\},\ U \in Si \qquad (1)$$

Y y es un vértice en Si que produce el L(v) mínimo. Si se hace un reemplazo, esto se debe al hecho de que se puede ir de v0 a v y recorrer una distancia más corta si se recorre un camino que incluye una arista (y,v).

Paso 3: Si cada vértice de Si (para algún $0 \leq i \leq n-2$) tiene la etiqueta (∞ , —) , entonces el grafo etiquetado contiene la información del camino más corto.

Si no, existe al menos un vértice v $\in$ Si que no está etiquetado como (∞ , —) y se realizan las siguientes tareas:

Se selecciona un vértice vi+1, tal que L(vi+1) sea mínimo. Puede haber varios de estos vértices cuyo caso se elige alguno de los posibles candidatos. El vértice vi+1 es un elemento de Si que es el más cercano a v0.

Se actualiza i= i+1.

Si i = n – 1, el grafo etiquetado contiene la información del camino más corto. Se regresa al paso 2.

3. Localización Montecarlo (MCL)

Localización Monte Carlo (MCL) es una técnica estadística para la localización del robot basada en la localización de Markov. Como en la localización de Markov, se requiere un mapa del entorno del Robot. Esto es un filtro de Bayes [10].

Localización Monte Carlo (MCL) fue desarrollado por Dellaert y Fox [11] Ellos tomaron el método de filtrado de partículas utilizado en otras áreas como visión por computadora. Localización Monte Carlo es uno de los métodos más populares en la localización, porque ofrece soluciones a una gran variedad de problemas.

La idea principal del filtro de Bayes es estimar una densidad de probabilidad sobre el estado espaciado condicionado sobre los datos. Este posterior que se suele llamar la creencia (Bel) y se denota

$$Bel(xt) = p(xt|\ d0...t) \qquad (2)$$

En la ecuación 2, x denota el estado, xt es el estado en el tiempo t y d0, t denota el tiempo inicial desde 0 a t. Denotando o (observación) y a (acción).

El modelo de movimiento, p(x‘|x,a), es una generalización probabilística de la cinemática del robot. Para un robot que operan en el plano las posiciones de x y x’ son variables en tres dimensiones.

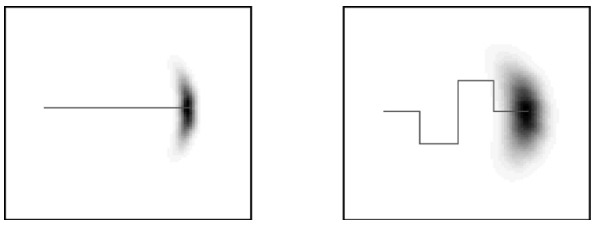

Fig. 1. Densidad p(x‘|x,a) después de mover 40 metros (diagrama izquierdo) y 80 metros (diagrama de la derecha).

En el ejemplo de la figura 1, la postura inicial de x se muestra a la izquierda, y la línea continua representa los datos de odometría, medido por el robot. El área sombreada de la derecha muestra la densidad p(x‘|x,a): la posición más oscura, es la que tiene mayor probabilidad. Una comparación de los dos diagramas revela que el margen de incertidumbre depende del movimiento global: A pesar de que la postura de un robot libre de ruido es la misma para ambos segmentos de movilidad, la incertidumbre en el diagrama de la derecha es más grande debido a la mayor distancia recorrida por el robot[11].

Un modelo de toma de muestras es una rutina que acepta x y a como entrada y genera al azar posiciones x’ distribuidas de acuerdo a p(x‘|x,a). La secuencia de conjuntos de partículas se aproxima a la densidad de un robot que mide sólo odometría.

4. Resultados Experimentales

Para poder modelar el comportamiento de este algoritmo: Dijkstra - Montecarlo, se construye el ambiente por el cual Robot tendrá que navegar. En éste laberinto de n * n (figura 2).el robot navegará desde el inicio(nodo 1) hasta alcanzar la meta (en este caso el nodo 25).

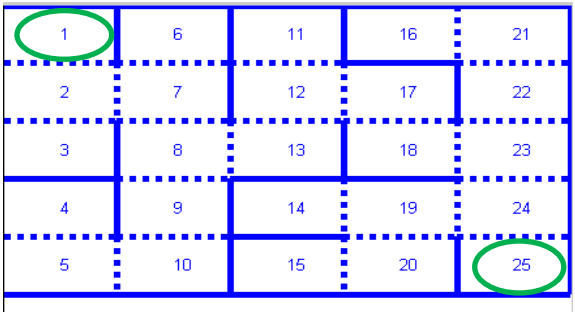

Fig. 2. Generación del Laberinto

En la Figura 2 se muestran las paredes (líneas solidas), mientras que las puertas por donde el Robot puede navegar hasta encontrar su objetivo se muestra mediante las líneas punteadas. Para encontrar el camino más corto dentro del laberinto, se utilizara el algoritmo de Dijkstra y la teoría de grafos. En la figura 3 se muestra el camino que siguió el robot navegando en el laberinto de la figura 2.

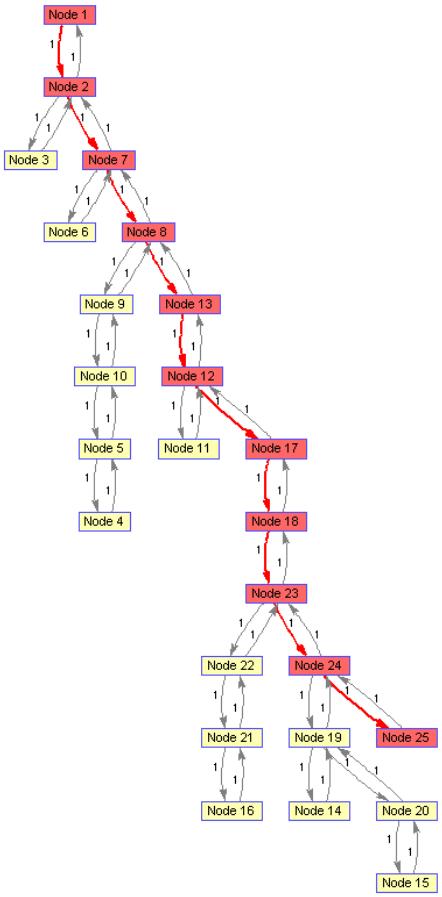

Fig. 3. Cálculo de la ruta mas corta mediante el algoritmo Dijkstra.

En el caso de la figura 3, la ruta más corta es; 1,2,7,8,13,12,17,18,23,24,25 . Una vez obtenida se traza la trayectoria por la cual navegara el Robot. La simulación se realiza por medio del algoritmo de Montecarlo. Esto se muestra en la figura 4.

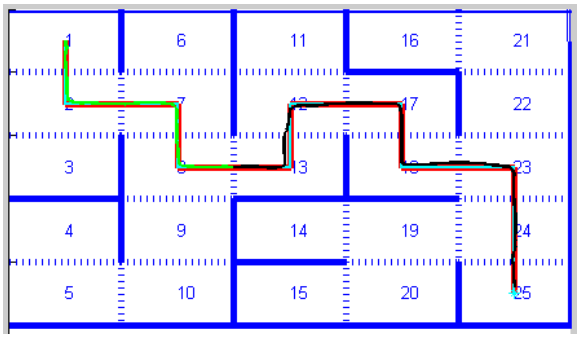

Fig. 4. Simulación de la navegación del Robot en una ruta establecida mediante Localización Monte Carlo.

Para un laberinto de 5 x 5 como el que se muestra en la figura 4, se realizaron diversos experimentos para demostrar que sin importar la posición de inicio o final, el algoritmo Dijkstra – MCL obtiene la ruta mas corto en un tiempo relativamente corto. Los tiempos para obtener la ruta mas corta y para recorrer la trayectoria se muestran en las tablas 1 y 2, respectivamente.

No. de Exp.	Tiempo
1	8.14E-04
2	9.62E-04
3	3.96E-04
4	4.68E-04
5	7.73E-04
6	1.16E-01
7	2.51E-04
8	2.47E-04
9	2.50E-04
10	2.52E-04

Tabla 1. Resultados experimentales de un laberinto de 5x5 para obtener la ruta mas corta.

Como puede verse en la tabla 1, los tiempos para poder calcular la ruta mas corta son consistentes, aunque depende el tiempo del número de nodos a recorrer. Además, se realizaron experimentos con laberintos de 6x6, 7x7 y 10x10, utilizando la metodología citada. La tabla 2 muestra los resultados de un laberinto 10x10.

No. de Exp.	Tiempo
1	2.94E+01
2	3.07E+01
3	9.04E+01
4	4.20E+01

5	1.22E+01
6	8.84E+01
7	1.52E+02
8	2.47E-04
9	3.56E+01
10	4.88E+01

Tabla 2. Resultados experimentales de un laberinto de 10x10 para obtener la ruta mas corta.

5. Conclusiones y trabajo futuro

Se ha mostrado que el algoritmo Dijkstra – MCL es útil para la planeación y optimización de rutas para robots móviles. Este algoritmo ha demostrado consistencia en diferentes escenarios de laberintos, por lo que puede ser usado en aplicaciones donde se necesita robustez y consistencia.
Como trabajo futuro se sugiere la implementación de este algoritmo en software embebido para implementarse en un robot móvil.

Referencias

[1] Tingle, D; Ball E; Augat, M., **Maze Solving by Learning State topologies**, Comp. Sci., Swarthmore College, 2009.

[2] Bruggemann, Bernd; Kamphans, Tom; Langetepe, Elmar, **Escaping from a Labyrinth with One-way Roads for Limited Robots**, Comp. Sci., University of Bonn, Bonn, Germany,2009.

[3] Rohrmuller, Florian; Althoff, Matthias; Wollherr, Dirk; Buss, Martin, **Probabilistic Mapping of Dynamic Obstacles Using Markov Chains for Replanning in Dynamic Environments**, IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, Sept, 22-26, 2008.

[4] Widodo Budiharto, Djoko Purwanto and Achmad Jazidie, **A robust obstacle avoidance for service robot using Bayesian approach**, International Journal of Advanced Robotic Systems, Vol. 8, No. 1, ISSN 1729-8806, pp 37-44, 2011.

[5] Ghahraman Zoubin, **An introduction to Hidden Markov Models and Bayesian**, International Journal of Pattern Recognition and Artificial Intelligence, no. 15, vol. 1, pp. 942, 2001.

[6] Rong-Jong Wai • Chia-Ming Liu • You-Wei Lin, **Robust path tracking control of mobile robot via dynamic petri recurrent fuzzy neural network**, Soft Comput., no 15, pp. 743–767, 2011.

[7] Yanrui Geng Æ Jinling Wang, **Adaptive estimation of multiple fading factors in Kalman filter for navigation applications**, GPS Solut., vol. 12, pp. 273–279, 2008.

[8] Xin Lu, Martin Camitz, **Finding the shortest paths by node combination**, Applied Mathematics and Computation, vol. 217, pp.6401–6408, 2011.

[9] Youming Li, Ardian Greca, and James Harris, **On Dijkstra's Algorithm for Deadlock Detection**, Advanced Techniques in Computing Sciences and Software Engineering, pp. 385-387, 2010.

[10] Stamenova, Stiliyana, **Solving the Maze: Robot Localization Using the Monte Carlo Localization Algorithm and Shape Context**, Macalester College, Ph.D. Thesis 2009.

[11] Thrun,Sebastian;Fox,Dieter;Burgard,Wolfram;Dellaert,Frank,**Robust Monte Carlo Localization for Mobile Robots**, Comp. Sci., Carnegie Mellon University ,Pittsburgh, PA,2001.